UIKit动力学

UIKit动力学是指可以给view添加满足动力学的一些效果，相比用animation来写，效率好很多。

UIDynamicAnimator

UIDynamicAnimator是为满足UIDynamicItem协议的元素提供力学相关效果，并为这些运动提供物理空间。
它是力学引擎和动态元素之间的媒介，这些元素通过物理行为UIDynamicBehavior添加。
构造方法：

```
-(instancetype)initWithReferenceView:(UIView *)view NS_DESIGNATED_INITIALIZER;
```

其中参数view是参考坐标系，也就是说动力行为参考的坐标，一般取为包含所有行为的子view的父view。

在一个界面上只需要一个UIDynamicAnimator就可以实现所有效果。
注意：在创建UIDynamicAnimator之后需要有一个对象对他持有，否则在arc模式下很快会被释放。

属性

```
// 引用视图 （只读）
@property (nullable, nonatomic, readonly) UIView *referenceView;
// 添加的行为数组 （只读）
@property (nonatomic, readonly, copy) NSArray<__kindof UIDynamicBehavior*> *behavi
ors;
// 是否正在运行 （只读）
@property (nonatomic, readonly, getter = isRunning) BOOL running;
// 运行以来的时间间隔（只读，注意编译条件）
#if UIKIT_DEFINE_AS_PROPERTIES
@property (nonatomic, readonly) NSTimeInterval elapsedTime;
#else
-(NSTimeInterval)elapsedTime;
#endif
// 代理
@property (nullable, nonatomic, weak) id <UIDynamicAnimatorDelegate> delegate;
```

方法

```
// 添加行为
-(void)addBehavior:(UIDynamicBehavior *)behavior;
// 移除行为
-(void)removeBehavior:(UIDynamicBehavior *)behavior;
// 移除所有行为
-(void)removeAllBehaviors;

// 返回指定矩形区域中的动态项目
-(NSArray<id<UIDynamicItem>> *)itemsInRect:(CGRect)rect;
// 在UIDynamicAnimator中更新我们已经修改的动态项目
-(void)updateItemUsingCurrentState:(id <UIDynamicItem>)item;
```

代理

```
@optional
// 当dynamicAnimator将要恢复调用
-(void)dynamicAnimatorWillResume:(UIDynamicAnimator *)animator;
// 当dynamicAnimator已经暂停调用
-(void)dynamicAnimatorDidPause:(UIDynamicAnimator *)animator;
```

UIDynamicBehavior

UIDynamicBehavior描述了相应的行为，具体使用的时候通常使用子类
UIGravityBehavior（重力）
UICollisionBehavior（碰撞）
UIAttachmentBehavior（吸附）
UIPushBehavior（推动）
UISnapBehavior（捕获）
UIDynamicItemBehavior

UIDynamicItem

UIDynamicItem是一个协议，只有实现该协议才能使用，目前UIView是实现该协议的。

```
@property (nonatomic, readwrite) CGPoint center;
@property (nonatomic, readonly) CGRect bounds;
@property (nonatomic, readwrite) CGAffineTransform transform;

@optional
/**
 The collision type represents how the dynamics system will evaluate collisions wi
th
 respect to the dynamic item. defaults to UIDynamicItemCollisionBoundsTypeRectangl
e
 */
@property (nonatomic, readonly) UIDynamicItemCollisionBoundsType collisionBoundsTy
pe NS_AVAILABLE_IOS(9_0);

/**
 The path must represent a convex polygon with counter clockwise winding and no se
lf intersection.
 The point (0,0) in the path corresponds to the dynamic item's center.
 */
@property (nonatomic, readonly) UIBezierPath *collisionBoundingPath NS_AVAILABLE_I
OS(9_0);
```

UIGravityBehavior

UIGravityBehavior用来模拟重力效果
示例：

```
UIGravityBehavior * gravity = [[UIGravityBehavior alloc] initWithItems:@[_hlView]]
;
CGVector gravityDircetion = {2,2};
gravity.angle = M_PI;
gravity.magnitude = 2;
[gravity setGravityDirection:gravityDircetion];
[_animator addBehavior:gravity];
```

gravityDircetion是重力的矢量，angle是重力的方向，magnitude是重力大小，也就是重力加速度大小。初始化传入的hlView就是需要实现重力效果的view

UICollisionBehavior

UICollisionBehavior模拟碰撞效果
示例：

```
UICollisionBehavior * collisionBehavior = [[UICollisionBehavior alloc] initWithIte
ms:@[_dynamicItem1View]];
collisionBehavior.translatesReferenceBoundsIntoBoundary = YES;
[_animator addBehavior:collisionBehavior];
```

这里初始化传入的是view数组。碰撞的类型有三种：

```
@property (nonatomic, readwrite) UICollisionBehaviorMode collisionMode;

typedef NS_OPTIONS(NSUInteger, UICollisionBehaviorMode) {
    // 元素之间的碰撞
    UICollisionBehaviorModeItems        = 1 << 0,
    // 边界碰撞
    UICollisionBehaviorModeBoundaries   = 1 << 1,
    // 碰撞所有
    UICollisionBehaviorModeEverything   = NSUIntegerMax
} NS_ENUM_AVAILABLE_IOS(7_0);
```

这里collisionBehavior.translatesReferenceBoundsIntoBoundary = YES;是用来将参考系的边界也能触发碰撞

```
-(void)addBoundaryWithIdentifier:(id <NSCopying>)identifier forPath:(UIBezierPath
*)bezierPath;
-(void)addBoundaryWithIdentifier:(id <NSCopying>)identifier fromPoint:(CGPoint)p1
toPoint:(CGPoint)p2;
-(nullable UIBezierPath *)boundaryWithIdentifier:(id <NSCopying>)identifier;
// 移除指定已命名的碰撞边界
-(void)removeBoundaryWithIdentifier:(id <NSCopying>)identifier;
// 获得所有命名
@property (nullable, nonatomic, readonly, copy) NSArray<id <NSCopying>> *boundaryI
dentifiers;
// 移除所有添加的碰撞边界
-(void)removeAllBoundaries;
```

我们也可以通过上面方法自己创建碰撞边界
UICollisionBehavior提供了相关的代理来处理碰撞相关的方法

```
@protocol UICollisionBehaviorDelegate <NSObject>
@optional
// 当一个两个动态元素之间发生碰撞时调用
- (void)collisionBehavior:(UICollisionBehavior *)behavior beganContactForItem:(id
<UIDynamicItem>)item1 withItem:(id <UIDynamicItem>)item2 atPoint:(CGPoint)p;
// 当一个两个动态元素之间碰撞结束时调用
- (void)collisionBehavior:(UICollisionBehavior *)behavior endedContactForItem:(id
<UIDynamicItem>)item1 withItem:(id <UIDynamicItem>)item2;

// 当一个动态元素与边界发生碰撞时调用
- (void)collisionBehavior:(UICollisionBehavior*)behavior beganContactForItem:(id <
UIDynamicItem>)item withBoundaryIdentifier:(nullable id <NSCopying>)identifier atP
oint:(CGPoint)p;
// 当一个动态元素与边界碰撞结束时调用
- (void)collisionBehavior:(UICollisionBehavior*)behavior endedContactForItem:(id <
UIDynamicItem>)item withBoundaryIdentifier:(nullable id <NSCopying>)identifier;
@end
```

UIAttachmentBehavior

UIAttachmentBehavior提供了元素之间，或者元素和锚点之间的吸附效果，效果分为两种：刚性、弹性，简单理解就是刚性就是两个物体之间用木棍等硬性物体连接，弹性就是两个物体之间用橡皮等弹性物体连接，两者之间的距离会发生弹性形变。

```
// 元素和锚点之间的吸附
- (instancetype)initWithItem:(id <UIDynamicItem>)item attachedToAnchor:(CGPoint)po
int;
// 元素和锚点之间的吸附，offset参数设置元素吸附力作用点的偏移量
- (instancetype)initWithItem:(id <UIDynamicItem>)item offsetFromCenter:(UIOffset)o
ffset attachedToAnchor:(CGPoint)point NS_DESIGNATED_INITIALIZER;

// 元素和元素之间的吸附
- (instancetype)initWithItem:(id <UIDynamicItem>)item1 attachedToItem:(id <UIDynam
icItem>)item2;
// 元素和元素之间的吸附，offset1、offset2分别是两个元素吸附力作用点的偏移量
- (instancetype)initWithItem:(id <UIDynamicItem>)item1 offsetFromCenter:(UIOffset)
offset1 attachedToItem:(id <UIDynamicItem>)item2 offsetFromCenter:(UIOffset)offset
2 NS_DESIGNATED_INITIALIZER;
```

```
@property (nonatomic, readonly, copy) NSArray<id <UIDynamicItem>> *items;

@property (readonly, nonatomic) UIAttachmentBehaviorType attachedBehaviorType;

@property (readwrite, nonatomic) CGPoint anchorPoint;

@property (readwrite, nonatomic) CGFloat length;
@property (readwrite, nonatomic) CGFloat damping; // 1: critical damping
@property (readwrite, nonatomic) CGFloat frequency; // in Hertz
@property (readwrite, nonatomic) CGFloat frictionTorque NS_AVAILABLE_IOS(9_0); //
default is 0.0
@property (readwrite, nonatomic) UIFloatRange attachmentRange NS_AVAILABLE_IOS(9_0
); // default is UIFloatRangeInfinite
```

属性中frequency表示振动频率，这个在刚性吸附时没有作用，弹性时会影响弹力，damping为阻力

UIPushBehavior

UIPushBehavior是给物体一个作用力，这个作用力可以是瞬发的也可以是持续的，简单理解就是有一阵风吹到物体上

```
UIPushBehavior * pushBehavior = [[UIPushBehavior alloc] initWithItems:@[_dynamicIt
em1View] mode:UIPushBehaviorModeInstantaneous];
[pushBehavior setPushDirection:CGVectorMake(0.5, 0.5)];
[pushBehavior setMagnitude:1.f];
[_animator addBehavior:pushBehavior];
```

示例代码如上

```
@property (nonatomic, readonly) UIPushBehaviorMode mode;
@property (nonatomic, readwrite) BOOL active;

@property (readwrite, nonatomic) CGFloat angle;
// A continuous force vector with a magnitude of 1.0, applied to a 100 point x 100
 point view whose density value is 1.0, results in view acceleration of 100 points
 per s^2
@property (readwrite, nonatomic) CGFloat magnitude;
@property (readwrite, nonatomic) CGVector pushDirection;
```

UISnapBehavior

UISnapBehavior是一个元素被一个点“捕获”，然后从起始位置移动到该点上，整个过程是有弹性效果的

```
 _snapBehavior = [[UISnapBehavior alloc] initWithItem:view1 snapToPoint:CGPointMak
e(50, 70)];
 [self.animator addBehavior:_snapBehavior];
```

在UISnapBehavior中可以设置damping来设置弹性效果

UIDynamicItemBehavior_other

UIDynamicItemBehavior视为是其他行为的一个特性上的补充，为其他的行为添加新的特性

```
@property (readwrite, nonatomic) CGFloat elasticity; // Usually between 0 (inelast
ic) and 1 (collide elastically)
@property (readwrite, nonatomic) CGFloat friction; // 0 being no friction between
objects slide along each other
@property (readwrite, nonatomic) CGFloat density; // 1 by default
@property (readwrite, nonatomic) CGFloat resistance; // 0: no velocity damping
@property (readwrite, nonatomic) CGFloat angularResistance; // 0: no angular veloc
ity damping

/*!
 Specifies the charge associated with the item behavior. Charge determines the deg
ree to which a dynamic item is affected by
 electric and magnetic fields. Note that this is a unitless quantity, it is up to
the developer to
 set charge and field strength appropriately. Defaults to 0.0
 */
@property (readwrite, nonatomic) CGFloat charge NS_AVAILABLE_IOS(9_0);

/*!
 If an item is anchored, it can participate in collisions, but will not exhibit
 any dynamic response. i.e. The item will behave more like a collision boundary.
 The default is NO
 */
@property (nonatomic, getter = isAnchored) BOOL anchored NS_AVAILABLE_IOS(9_0);

@property (readwrite, nonatomic) BOOL allowsRotation; // force an item to never ro
tate
```

属性包括，弹性，摩擦，密度，线 / 角速度阻尼，电荷，是否可旋转等。

后续

除了官方提供的之外后续可以有两个发展，一个是封装使用，在实际的应用中很多东西都是成对出现的，我们可以自己封装一个UIDynamicBehavior的子类，传入相应的item对象，然后内部添加想要的行为。另一种是自定义，自己定义想要的规则和相应的行为（不存在的）