# 第三章 Common模块

原文地址：<http://ohcoder.com/blog/2012/12/02/common/>。

## 3.1 关于

Common模块包含了设置、内存管理和矢量数学。

## 3.2 设置

b2Settings.h头文件包含：

* int32和float32等数据类型
* 常量
* 分配器包装（Allocation wrappers）
* 版本号

### 数据类型（Types）

Box2D定义了float32，int8等类型，以便能方便地定义结构的大小。

### 常量

Box2D定义了一些常量。这些都被记录在b2Settings.h头文件中。正常情况下不需要调整这些常量。

Box2D使用浮点数来进行物理模拟和碰撞计算。由于数据舍入误差的原因，需要定义一些公差。其中一些公差是绝对的，一些公差是相对的。绝对的公差使用MKS（Meters-Kilometers-Seconds）单位。

### 分配器包装（Allocation wrappers）

设置文件中定义了b2Alloc和b2Free两个方法，用于大内存分配。你可以使用这两个方法来调用你自己的内存管理系统。

### 版本号

b2Version结构保存了当前版本，你可以在运行时查询当前的版本号。

## 3.3 内存管理（Memory Management）

Box2D的许多设计决策都是为了能快速有效地使用内存。在本节我将论述Box2D如何和为什么要分配内存。

Box2D倾向于分配大量的小对象（50～300字节左右）。这样通过malloc或new在系统的堆（heap）上分配内存太低效，并且容易产生内存碎片。多数这些小型对象的生命期都很短暂，例如触点（contact），可能会维持几个时间步长。所以我们需要为这些对象提供一个有效的分配器（allocator）。

Box2D的解决方案是使用小型对象分配器（small object allocator，SOA），SOA维护了许多不定尺寸的可生长的内存池（growable pool）。当有内存分配请求时，SOA会返回一块最匹配的内存。当内存块释放掉以后，它会回到池中。这些操作都十分快速，导致很小的堆流量。

因为Box2D使用了SOA，所以你应该永远也不必去new或malloc物体，定制器或关节。你只需分配一个b2World，它为你提供了创建物体，定制器和关节的工厂（factory）。这使得Box2D可以使用SOA并且将细节隐藏起来。永远也不要去delete或free一个物体，定制器或关节。

当执行一个时间步长的时候，Box2D会需要一些临时内存。为此，它通过调用b2StackAllocator，使用了一个栈（stack）分配器来消除单步堆分配。你不需要关心栈分配器，但在此作一个了解还是不错的。

## 3.4 数学

Box2D包含了一个简单的小巧的向量和矩阵模块。这部分设计用来满足Box2D内部使用和API接口的需要。这部分所有成员都是公开的，所以你可以在你的应用程序中自由的使用。

数学库尽量保持着简单，保持着Box2D的可移植性和可维护性。