——中文帮助文档 V1.0

版权信息

本版本为 Impact 中文文档 1.0 版本，有很多不足之处将在稍后的更新中修改。欢迎大家批评指正。

如果有兴趣再发行或再版本手册的全部或部分内容，或者有任何问题，请联系版权所有者 ： coolcodestudio@163.com。

目录

前言

Impact 是一个 JavaScript 游戏引擎，可以为桌面和手机浏览器开发令人惊叹的 HTML5 游戏。

Impact 可以在所有支持 HTML5 的浏览器上运行：火狐、Chrome、Safari、Opera，甚至是 Internet Explorer 9 。当然也包括 iPhone、iPod Touch 和 iPad。

希望大家积极分享 html5 移动手机游戏的开发经验，让我们共同提高。

感谢我的合作伙伴 coolcode 大牛对我学习 html5 的帮助。

欢迎大家关注：www.coolcode.org（coolcode 大牛的网站）

www.lywmobile.com

有问题可以 QQ 联系：2247921616

我们将尽力解答您的问题。

作　　者

本手册由 CoolCode-Studio 制作。

主题

安装与入门

安装

尽管用 Impact 写的游戏可以脱机工作，但大多数浏览器都不允许。这些浏览器由于同源策略的缺陷，拒绝访问某些功能（在 `file://game/lib/` 下的文件与在 `file://game/` 下的文件不被认为是在同一个“域”下）。 Opera 是个显著的例外。

此外，Weltmeister 关卡编辑器使用一些 .php 脚本来加载和保存关卡，列出目录的内容。

长话短说，您需要一个网页服务器。说到网页服务器，并不意味着是一个单独的服务器计算机，它只是你电脑里的一个提供 web 服务的程序。如果您使用的是 MacOSX ，您就已经拥有网页服务器了，也安装了 PHP ，只需启用它。在 Windows 中，您可以安装 Apache 和 PHP。这不是很复杂。如果您使用的是 Linux，那么您应该是个电脑虫，完全可以自己配置出来。^_^

如果你想手动安装 Apache 和 PHP，需要帮助，请问谷歌。

现在，你已经有网页服务器了，PHP 也运行了，对吧？好！不仅要把 Impact 解压到网页服务器的根目录下，还要让你的浏览器指向 http://localhost/ 。

您应该看到了一个目录列表，包括您刚刚创建的 impact/ 子目录。试着通过 http://localhost/impact/weltmeister.html 加载 Weltmeister 。载入后应该没有任何错误。

如果有错误信息，你应该在浏览器控制台里查看。如果你不知道去哪里找错误控制台，往下读。

其他解决方案：

如果你不想使用 PHP 或者 Apache，Conner Petzold 制作了一个 nodejs 模块，它可以让 Impact 在一个 HTTP 服务器节点上运行。他的 node-impact 模块在 GitHub 上有。

如果您使用的是 Windows，并且喜欢在 IIS .NET 上开发 Impact，试试 Mike Hamilton 的 ImpactJS-IIS-.NET-API 项目。

为方便 Ruby 爱好者，Chris Darroch 为 Impact 集成了 Sinatra 后台，您只需在 lib/weltmeister/config.js 的 API 调用中删除 .php 扩展，启用 impact.rb。

设置工作环境

如果你正在用 Impact 制作游戏（或者其它浏览器相关），浏览器的开发工具和 JavaScript 控制台将派上用场。如果你不使用它们，某些东西不工作了，你将陷入茫然。

在谷歌浏览器中，你可以点击“扳手”图标 -> 工具 -> JavaScript 控制台，找到控制台。

在 Safari 浏览器中，你必须先开启开发菜单：打开 Safari 的 Preferences through Safari -> Preferences，点击“高级”选项卡，选中“Show Develop menu in menu bar”。之后，您可以通过 Develop -> Show Error Console 打开控制台。

对于 Firefox，请安装最棒的 Firebug 扩展。

在 Opera 中，你可以在菜单 -> 网页 ->开发者工具 -> Opera Dragonfly 里找到开发工具。

如果您正在使用 Internet Explorer ，那就考虑使用其他浏览器吧。Impact 可以在 IE9 上运行，但是速度有点慢。您可以按 F12 键打开”开发人员工具“。

现在你只需要一个文本编辑器来编辑你的源文件。如果你是自虐型的，Windows 的记事本都可以。然而，使用支持语法高亮的编辑器将使您生活得更轻松。

我使用的是 Komodo Edit，基本上是 Komodo IDE 的免费版本。不用看它的名字，Komodo Edit 还是一种 IDE。它真的很棒，可以用在 Windows、MacOSX 和 Linux 上。

如果你只是想要一个轻量级的编辑器，我可以推荐 SciTE for Windows 和 Linux。它还可以用在 MacOSX 上，但是设置它有点麻烦。

对于 MacOSX ，听说 BBEdit 和 TextMate 大概是“最好”的编辑器。

这些只是根据我个人的经验提出的一些建议。不要信任我。请随意到处看看，尝试不同的编辑器和 IDE。

Impact 安装完成以后，应该有以下的目录结构：

```
media/
lib/
lib/game/
lib/game/entities/
lib/game/levels/
lib/impact/
lib/weltmeister/
```

您的所有游戏资源，比如图像、声音和音乐，都应放在 `media/` 目录下。

`lib/` 目录存放所有的 JavaScript 文件。`lib/impact/` 存放 Impact 引擎本身。

`lib/weltmeister/` 存放关卡编辑器的所有源文件。

您自己的游戏源文件应该放在 `lib/game/` 下。

实体的源文件应该放在 `lib/game/entities/`下。

这样 Weltmeister 才能找到它们。

HTML 文件

用 Impact 写的游戏可以直接在浏览器中运行。不需要任何插件。然而，由于 JavaScript 本身没有显示东西的地方，Impact 需要在一个 HTML 页的 `<canvas>` 标记上渲染游戏画面。

最基本的 HTML 页看起来类似这样：

```
<!DOCTYPE html>
<html>
<head>
	<title>My Awesome Game!</title>
	<script type="text/javascript" src="lib/impact/impact.js"></script>
	<script type="text/javascript" src="lib/game/game.js"></script>
</head>
<body>
	<canvas id="canvas"></canvas>
</body>
</html>
```

当然，这个页可以用后来的内容（文字、链接、图像等）进一步扩展，就像其他 HTML 页一样。也可以使用 CSS 样式。我在这儿真的不能再详细说了，因为这本身就是一个庞大的话题。但是，这个基本的 HTML 页确实是你的游戏开发工作所需要的全部。

请注意，这个 HTML 页只引用了两个 JavaScript 文件：引擎和游戏主脚本。其他的所有 JavaScript 文件会被这两个文件自动包含进来。

在发布您的游戏之前，也可以把所有的 JavaScript 烘焙到一个（压缩的）文件中。这将大大缩短你的游戏的初始加载时间。然而，为方便开发，最好是让文件分开，有一个更好地综览。更多信息，请参阅烘焙。

您的游戏所需的所有图像和声音文件，也将通过 JavaScript 动态加载。Impact 的预加载，确保在游戏开始之前，所有的资源都被加载进来。

Impact 自带一个默认的 `index.html` ，用来加载 `lib/game/game.js` 文件。

`weltmeister.html` 用来加载编辑器。更多信息，请查看 Weltmeister 编辑器的介绍。

模块

JavaScript 本身并不提供 `include()` 函数，来加载其他 JavaScript 源文件。你可以写你自己的 `include()` 函数，用 Ajax 来加载文件，但是这会使你的游戏无法调试，因为所有的行号和文件名都会丢失，错误信息将比“匿名函数 anonymous() 中的错误“更难确定。

相反，Impact 的源代码被组织成模块。一个模块定义通常看起来类似这样：

```
ig.module(
    'game.my-file'
)
.requires(
    'impact.game',
```

```
    'impact.image',
    'game.other-file'
)
.defines(function(){


    //这个模块的代码
});
```

模块名“game.my-file”直接对应文件名。因此，这个模块位于 lib/game/my-file.js。同样，在 requires() 方法中列出的模块将分别从 lib/impact/game.js、 lib/impact/image.js 和 lib/game/other-file.js 载入。这些必需的文件会在模块主体（传递给 .defines() 的函数）执行之前被加载。你可以阅读更多”ig 核心“参考资料中有关模块的部分。

Impact 是如何工作的？

Impact 引擎的核心，不是一个库，而是一个框架。也就是说，Impact 提供一个功能齐全的盒子，你可以把你的代码扔进去。Impact 运行其自身。你只需要把你的东西添加给它，也由引擎来管理。

“你的东西”，在大多数情况下，是 Impact 的一个基类的子类。最重要的一个就是 ig.Entity 类。游戏世界中的每一个对象，都将是 ig.Entity 的一个子类。

只要你启动你的游戏，Impact 就会建立一个调用 ig.system.run() 方法的时

间间隔，每秒钟 60 次。这个方法做一些看家的东西，然后调用你的游戏的 run() 方法（默认情况下，只调用它自身的 update() 和 draw() ）。

ig.Game 的 draw() 方法是很无聊的，它只是清除屏幕，在每一个背景层和实体上调用 draw()。

update() 方法更新背景层的位置（这是它感兴趣的地方），在每个实体上调用 update() 。实体默认的 update() 方法，是根据它的物理属性（位置、速度、反弹力等）移动，并且把游戏的碰撞地图也考虑进去。

在所有的实体更新完成后，游戏的 .checkEntities() 方法被调用。这个用来解决所有的动态碰撞，也就是实体与实体的碰撞。如果它与另一个实体重叠，而且“希望”检查，它还调用一个实体的 .check() 方法（更多详细信息，请参阅”类引用“）。

您可以在自己的 ig.Entity 和 ig.Game 子类中重写这些方法—您可以提供自己的逻辑，然后，如果你喜欢，可以用 this.parent() 调用原有方法。

记住，所有这一切都发生在每一帧上。那就是（如果浏览器能够跟上的话）每秒 60 次。

Weltmeister

本文将教你 Weltmeister Level Editor 的基本知识。请注意，还有一个可用的视频教程。

启动 Weltmeister

如果您的 Web 服务器配置正确，你应该能够将浏览器指向 http://localhost/impact/weltmeister.html，/impact/ 当然是你解压 Impact 的目录。

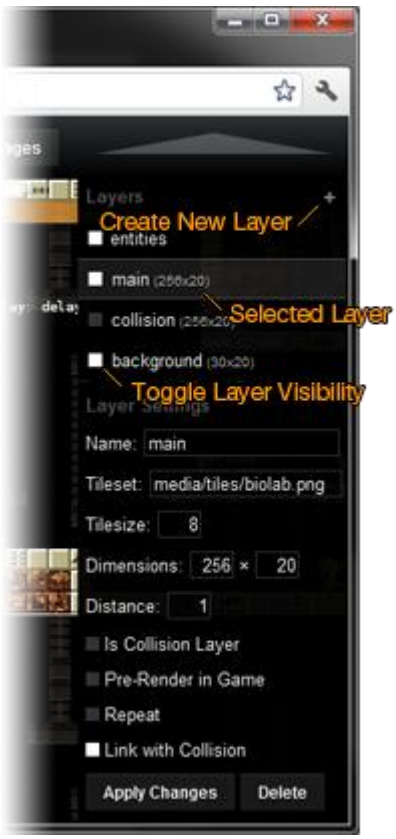
当 Weltmeister 载入完成，一个新的、空的关卡就呈现了。如果它没有加载，请检查 JavaScript 错误控制台，并确保您的服务器上的 PHP 工作正常。

控制

- 按住**鼠标右键**，或按住 **Ctrl 键**拖拽周围视图。
- **鼠标左键**绘制当前选定的图块，或者，当实体层被选中时，选择光标下的实体。
- **空格键**用来切换图块的选择 ,或切换实体选择菜单 ,取决于哪个层被激活。
- **Backspace 键**或 **Del 键**用来删除当前选定的实体。**C 键**用来复制实体。
- **Z 键**和 **Y 键**分别为”撤消“和”重做“。
- 按 **G 键**切换 BackgroundMaps 网格。
- 按住 **Shift 键**和**鼠标左键**，选择地图（或图块选择屏）的一个区域，作为新的”刷子“。

你可以在文件 lib/weltmeister/config.js 中更改这些控制。

使用层



你会看到，即使是在您的空地图上，也出现了一个层—实体层。你的关卡的所有实体都放在这里。这一层是永远存在的，无法删除。

当你点击“+”图标时，一个新层就创建了，并且是被选中的。您可以通过点击菜单上的层，或按键盘上相应的数字键，来选择层。

点击层名前面的复选框，或按下 Shift 键与相应的数字键，可以切换层的可见性。这个可见性只在编辑器内有效，在你的游戏中没有任何效果（如果您使用默认的 .loadLevel() 方法）。

您将看到被选中层的 Layer Settings 菜单（如果它不是实体层）。您可以在这儿更改这个层的 Name 和 Tileset，以及 Tilesize 和该层中图块的宽度和高度。

改变层的 Dimensions 或 Tilesize 时要小心—没有”撤消“按钮！

层的 Distance 决定了该层相对于游戏屏幕滚动的快慢。如果是 1 个距离，则层与屏幕同步滚动。如果是 2 个距离，则地图的滚动速度减半，使得背景看上去好像很远。这只在次要的背景层有用。主层的话，应该始终为 1 。

Repeat 复选框规定了该层是否以重复的“背景模式”绘制。同样，这在次要的背景层有用。

如果选中了 Link with Collision ,则对该层的所有操作 ,都将应用到碰撞层。这就意味着，你在该层上绘制的每一个图块，同样会应用在碰撞层(tile #1)。你每删除一个图块，同样会从碰撞层删除。

您可以像创建其他层一样创建一个碰撞层，然后给它命名为 collision 即可。它会自动变成这个关卡的碰撞层。这也意味着 ,这一层不会在你的游戏中画出来。

绘图

当背景层或碰撞层被选中时，您可以按**空格键**打开图块选择菜单。左键单击选择一个图块。选择一个图块后，简单的左键单击来放置它。按住 **Shift 键**，并单击+拖拽，可一次选择多个图块。

如果你想从该层删除一些图块，则按**空格键**，然后在图块选择菜单**以外**的任何地方单击。这将选中“空”的图块。再次左键单击你想要放置的地方。

所有的图块绘制操作，都可以使用 **Z 键**或 **Y 键**进行“撤消”或“重做”。

载入您的实体

在实体层被选中后，您可以通过按空格键放置一个实体，并从菜单中选择它。

这个菜单将会列出在 lib/game/entities/ 目录（这个路径可以在

lib/weltmeister/config.js 中更改）下找到的所有实体。实体名必须按规定的格式书写，实体才能被 Weltmeister 载入。

例如，文件 lib/game/entities/player.js 中必须定义一个名为 EntityPlayer 的 ig.Entity 子类。文件名中的连接线”-“是实体名中大写字母的标志。例如，文件 enemy-blob.js 中定义的实体名必须为 EntityEnemyBlob 。

如果文件名和实体名不匹配，Weltmeister 会在启动时告诉你的。

实体可以有一些仅在 Weltmeister 使用的特殊属性。这些属性都以 _wm 开头。查看 ig.Entity 的详细信息。

操作实体

实体可以通过左键拖拽控制他们。如果实体是可伸缩的(._wmScalable = true)，你可以通过拖拽边缘改变它的大小。

实体可以用 Backspace 键或 Del 键删除。您可以按下 C 键复制实体。注意，这只是创建一个相同类型的新实体，它不会复制原始实体的所有属性。

当一个实体被选中时，你会在右侧看到 Entity Settings 菜单。某些设置，如 x、y、size.x 和 size.y ，当您拖动实体或改变其大小时，将会自动更新。

如果你点击一个属性，“键”(name)以及它的”值“就会出现在下方的输入框内。如果在“Value”框中按回车，这个值就被写入该实体。

您可以输入一个新的”键“和”值“，来给实体添加任意数量的设置。例如，如果您想给实体设置一个名称属性“myName”，只需输入“name”作为 Key，“myName”作为 Value，然后按 Enter 键。您可以通过指定一个空字符串作为”值“，来删除属性。

您可以把两个实体连接起来，给其中一个实体一个”名字“，给另一个实体一个同名"目标"。例如：

```
EntityDoor
  key: name
value: door1


EntityTrigger
  key: target.1
value: door1
```

一个实体可以有多个目标，所以有 target.1、target.2 等等这样的目标编号。如果两个实体连接起来了，将在它们之间画一条线。

然后，您可以这样使用这个 trigger 实体的目标对象：

```
EntityTrigger = ig.Entity.extend({
    size: {x: 16, y: 16},
    target: {},
    checkAgainst: ig.Entity.TYPE.BOTH,


    _wmScalable: true,
    _wmDrawBox: true,
    _wmBoxColor: 'rgba(196, 255, 0, 0.7)',
```

```
    check: function( other ) {
        // Iterate over all targets
        for( var t in this.target ) {
            var ent = ig.game.getEntityByName( this.target[t] );

            // Check if we got a "door" entity with the given name
            if( ent && ent instanceof EntityDoor ) {
                ent.open();
            }
        }
    }
});
```

所有的 Entity Settings 都将被直接写入实体的属性，所以一定要小心，不要覆盖你需要的属性，比如 draw 或 update 函数。

使用资源

图像和声音文件所在路径

加载图像或声音文件时，这个文件的路径总是相对于游戏运行的 .html 文件的。换句话说，你的 .js 文件的位置是无关紧要的。

这样，让我们假设有以下的目录结构，你的 game.js 由 game.html 加载：

```
/mygame/game.html
/mygame/lib/game.js
/mygame/media/some-image.png
/mygame/media/some-sound.ogg
/mygame/media/some-sound.mp3
```

现在，您想要在 game.js 文件中加载 some-image.png，需要像下面这样做：

```
// The game.html file is in /mygame/, that means that all
// assets can be referenced relative to /mygame/
var someImage = new ig.Image( 'media/some-image.png' );

// It works exactly the same for sounds:
var someSound = new ig.Sound( 'media/some-sound.*' );

// Or AnimationSheets
var someAnimSheet = new ig.AnimationSheet( 'media/some-image.png', 8, 8 );
```

路径 “media/some-sound.*“ 实际上是有效的。Impact 将根据浏览器的能力来加载相应的声音文件（MP3 或 Ogg 格式）。见 ig.Sound。

使用预加载器

你的游戏所需要的所有声音和图像文件，都应该通过”预加载器“来载入，以便在需要时使它们真正可用。

在一个尚未加载的图像上调用 draw() 方法，它不会做任何事情。试图播放一个尚未加载的声音文件，会导致严重的滞后。

在加载期间创建的所有 ig.Image、ig.Font、ig.AnimationSheet 和 ig.Sound 实例都将被追加到 Preloader 的资源链。只有在运行时创建的图像和声音，才不会由 Preloader 载入。

```
MyGame = ig.Game.extend({
    // This image will be loaded by the preloader. It is created
```

```
    // as soon as the script is executed
    titleImage: new ig.Image( 'media/title.png' ),

    init: function() {
        // This image file will NOT be loaded by the preloader. The
        // init() method is only called after the preloader finishes
        // and the game is started.
        this.backgroundImage = new ig.Image( 'media/background.png' );
    }
});
```

缓存

所有资源均自动缓存。不管多长时间需要它们，它们只加载一次。例如，如果两个实体共享同一个 AnimationSheet 图像，它不会被加载两次。因此，这是完全有效的：

```
EntityPlayer = ig.Entity.extend({
    animSheet: new ig.AnimationSheet( 'media/player.png', 16, 16 )
    …
});

EntityPlayerGibs = ig.Entity.extend({
    // The player gibs are in the same image file
    animSheet: new ig.AnimationSheet( 'media/player.png', 8, 8 )
    …
});
```

您也可以用这个办法来确保背景音乐由 Preloader 加载。在加载时，你不能使用 ig.music.add() ,但是你可以使用已经加载的 ig.Sound 的实例的声音文件。

```
MyGame = ig.Game.extend({

    // Make sure the music is loaded by the preloader.
    // Passing 'false' as the second argument, prevents loading this
    // sound as multi channel.
    bgtune: new ig.Sound( 'media/background-tune.*', false ),


    init: function() {
        // Now add the file to the playlist
        ig.music.add( this.bgtune );


        // You could also just specify the path again:
        // ig.music.add( 'media/background-tune.*' );


        // Ready to Rock!
        ig.music.play();
    }
});
```

碰撞

概述

Impact 中的碰撞检测由两个单独的步骤来完成：“静态”碰撞（实体与世界）和“动态”碰撞（实体与实体）。

静态碰撞是实体在移动过程中自己解决的，也就是说，实体检测是否可以移动到

它想要去的地方。在实体移动之前，它做了一个从当前位置到目的地的追踪。游戏的 CollisionMap 用追踪结果回应—是否以及哪里有一个碰撞。

动态碰撞是在所有实体都移动以后解决的。动态碰撞只检查当前两个实体是否是重叠的—如果是的话，就把它们移开。

重要的是，要注意，静态碰撞始终是准确的、帧率独立的，而动态碰撞却不是这样的。这就意味着，快速移动的实体不能穿墙移动，但是两个快速移动的实体之间的碰撞可能被忽视。

Impact 始终保持最低每秒 20 帧的帧率，以使所有的动态碰撞仍然足够准确。见 ig.Timer.maxStep。

静态碰撞

默认情况下，所有的实体都会与游戏的 .collisionMap 碰撞。实体的 .update() 方法根据实体的属性计算移动速度，然后对 CollisionMap 做一个追踪。然后，追踪的结果由实体的 .handleMovementTrace() 方法处理。

所以，如果你不想让实体参与静态碰撞，最简单的办法，就是像下面这样为该实体重写 .handleMovementTrace() 方法：

```
handleMovementTrace: function( res ) {
    // This completely ignores the trace result (res) and always
    // moves the entity according to its velocity
    this.pos.x += this.vel.x * ig.system.tick;
    this.pos.y += this.vel.y * ig.system.tick;
```

```
},
```

您也可以检测实体与哪个图块碰撞了，并采取相应的行动（例如，是否有光滑的图块）。

如果你想在玩家实体以一定的速度落地时播放一段声音，你可以重写 .handleMovementTrace() ，检查碰撞和速度。

```
EntityPlayer = ig.Entity.extend({

…

handleMovementTrace: function( res ) {
    if( res.collision.y && this.vel.y > 32 ) {
        this.soundBounce.play();
    }

    // Continue resolving the collision as normal
    this.parent(res);
},

…

});
```

您可以使用 Weltmeister 创建 CollisionMap。默认情况下，游戏的 .loadLevel() 方法假定一个名为“collision”的层为关卡的 CollisionMap。

您可以随时手动创建一个 CollisionMap：

```
//在你的游戏的 init() 方法里

var map = [
    0,0,0,0
    0,1,1,0
    0,0,0,0
];
this.collisionMap = new ig.CollisionMap( 8, map );
```

如果你不指定 CollisionMap，将使用一个空的虚拟层（ig.CollisionMap.staticNoCollision）。

动态碰撞

实体与实体的碰撞有一点复杂，因为一个实体可以有多种碰撞模式。如果两个实体相互碰撞，则它们两个的碰撞模式决定了是否以及如何解决碰撞。

实体的碰撞模式是由它的 .collides 属性决定的。默认情况下，该属性设置为 ig.Entity.COLLIDES.NEVER，忽略所有的碰撞，不管其他实体拥有哪种碰撞模式。

其他的碰撞模式有 LITE、 PASSIVE、ACTIVE 和 FIXED。

如果一个 PASSIVE 实体与一个 ACTIVE 实体碰撞，则分开两个实体。ACTIVE 与 ACTIVE 碰撞的情况也是如此。

PASSIVE 模式存在的原因，是 PASSIVE 与 PASSIVE 碰撞根本不会被解决（换言之，他们只会重叠）。这可是有用的，敌人能够“穿过”对方，而不是挡住道路。

这些 PASSIVE 实体仍然会与 ACTIVE 实体碰撞。

其他两个模式实际上是为相互碰撞定义了一个“弱”实体和一个“强”实体。在这种情况下，只移动“弱”实体来解决碰撞。

设置为 LITE 碰撞模式的实体始终是“弱”实体，也就是移动的实体。设置为 FIXED 碰撞模式的实体始终是“强”实体，它就待在那里。

LITE 实体只和 ACTIVE 或 FIXED 实体碰撞。FIXED 实体会与 LITE、PASSIVE 和 ACTIVE 实体碰撞。

还迷惑吗？下面是一些常规建议：

玩家和敌人角色通常应该是 PASSIVE 的，以使他们不互相碰撞，但是可以与 ACTIVE 实体（如箱子）碰撞。

平台和电梯应该是 FIXED 的，以使他们不用理会是否有其它实体挡住了它们的路。

所有纯视觉的、不影响游戏的实体，都应该为 Lite 或 NEVER。

动画

创建动画

Impact 中的动画由 AnimationSheet（一个大的包含所有单个动画帧的图像）来处理。

AnimationSheet 帧在左上角从 0 开始编号，然后继续读取方向。请注意，这与 BackgroundMaps 的图块从 1 开始编号不同。

要创建一个 AnimationSheet，你得指定要使用的图像文件，以及这个图像的每一帧的宽度和高度。这个 AnimationSheet 应作为由预加载器加载的一个类的属性来建立。更多信息，请参阅“使用资源”。

当你创建了一个新的 Animation，你就可以交出 AnimationSheet 了。每个帧的显示时间是以“秒”来计算的。一组帧规定了动画序列。

```
// Create an AnimationSheet with the blob.png file.
// Each frame is 16x16 px
var animSheet = new ig.AnimationSheet( 'blob.png', 16, 16 );

// Create an animation using the AnimationSheet.
// Display each frame for 0.2 Seconds. The Sequence
// is [2,3,4] and the Animation stops at the last
// frame (last parameter is true)
var jumpAnim = new ig.Animation( animSheet, 0.2, [2,3,4], true );
```

创建动画以后，你必须使用 .update() 方法为每个游戏帧更新它。你可以使用 .draw() 方法绘制它。

Entities 动画

在让实体产生动画的时候 ,你可以使用一些实用的实体方法 ,让自己更轻松一些。

每个实体都有一个 .animSheet 属性，你可以在这里为它加载默认的 AnimationSheet。

实体还有一个 .anims 对象，你用 .addAnim() 方法定义的所有动画都添加给它了。和 Animations 的构造函数一样，.addAnim() 方法采用同样的参数，除非它总是用 .animSheet 作为 AnimationSheet。

实体的 .draw() 方法在实体的位置自动绘制 .currentAnim。

```
// Create your own entity, subclassed from ig.Enitity
EntityBlob = ig.Entity.extend({
    // Load an animation sheet
    animSheet: new ig.AnimationSheet( 'blob.png', 16, 16 ),

    init: function( x, y, settings ) {
        // Add animations for the animation sheet
        this.addAnim( 'idle', 1.5, [1,1,2] );
        this.addAnim( 'jump', 0.2, [2,3,4], true );

        // Call the parent constructor
        this.parent( x, y, settings );
    }
});
```

这个实体已经可以用在 Weltmeister 上了，会在游戏中动起来。它的 .currentAnim 将被设置为“idle”动画，因为它是最先添加的。

若要切换动画，您只要把 .currentAnim 设置为在 .anims 中定义的一个动画即可。例如：

```
update: function() {
    if( someCondition ) {
        this.currentAnim = this.anims.jump.rewind();
    }
    else {
        this.currentAnim = this.anims.idle;
    }
}
```

BackgroundMaps 动画

对于 BackgroundMaps，您可以指定哪些图块为“动画”。每个 BackgroundMap 都有一个 .anims 属性，为 Animation 画出了图块的编号。例如：

```
var data = [
    [1,2,6],
    [0,3,5],
    [2,8,1],
];
var bg = new ig.BackgroundMap( 16, data, 'media/tileset.png' );

// The image used for the AnimationSheet *can* be different from
// the one used as the tileset for the BackgroundMap
var as = new ig.AnimationSheet( 'media/tiles.png', 16, 16 );
```

```
// Sets tile number 5 of the BackgroundMap to be animated
bg.anims[5] = new ig.Animation( as, 0.1, [0,1,2,3,4] );
```

请注意， Animations 不是由 BackgroundMap 来更新的，而是必须由外部来更新的。ig.Game 有一些工具可以做这个。见 .backgroundAnims。

手机平台上的 Impact

概述

尽管 Impact 在大多数手机平台上运行得出奇地好，还是有一些问题是需要注意的：

性能—手机平台仍然相当缓慢，但是使用一些技巧，差不多总是可以达到良好的帧速率的。

分辨率—许多设备的屏幕大小不同。 iPhone 4 的设备像素的比例进一步复杂化的东西。

声音—在一个时间只有一个声音，可以起到和加载的声音文件有浏览器的 bug。

多点触控问题—可悲的是，这个 IOS4.2.1 介绍一个在浏览器中的第二次触摸有时无法识别的错误。苹果已经意识到了这个问题。

现在，发挥 HTML5 的游戏是最好的移动设备的 iPhone 3GS 或第三代的 iPod Touch。影响，也对所有其他的 IOS 设备运行良好，但这些都是有点慢。即使是较新的 iPhone 4 和 iPad 无法与之抗衡 3GS 的速度，因为他们有更多的像素画。

性能 – – 移动平台仍然是相当缓慢，但一些技巧，您可以实现几乎总是好的帧速率决议 – 屏幕大小是不同的许多设备。IPhone 4 的设备像素比进一步复杂化的东西的声音 – – 可以播放一次只有一个声音，有浏览器的加载声音文件的 bug。多点触摸的问题 – – 遗憾的是，此 iOS 4.2.1 推出凡有时会在浏览器中的第二个触摸不能识别的 bug。苹果已经意识到这个问题。

现在是最好的移动设备上播放 HTML5 游戏是 iPhone 3GS 或第 3 代 iPod Touch。影响还在井上所有其他的 iOS 设备运行，但那些稍微慢一些。甚至较新的 iPhone 4 和 iPad 无法与竞争 3GS 的速度，因为他们有更多像素画。

我的猜测是，苹果已经完全硬件加速的 HTML5 画布标记某个地方在他们的路线图 iOS，这样大的屏幕尺寸，iPhone 4 和 iPad 的再也不重要了。它只会变得更快。

针对不同的设备

要检查的特定设备，可以使用 ig.ua.* 属性的影响。尽快加载主 impact.js 文件，可以使用 ig.ua。

这一点，可以，例如，游戏开始时你不同的分辨率，取决于其运行的设备：

```
if( ig.ua.mobile ) {
    // Disable sound for all mobile devices
    ig.Sound.enabled = false;
}

if( ig.ua.iPhone4 ) {
    // The iPhone 4 has more pixels - we'll scale the
```

```
    // game up by a factor of 4
    ig.main('#canvas', MyGame, 60, 160, 160, 4);
}
else if( ig.ua.mobile ) {
    // All other mobile devices
    ig.main('#canvas', MyGame, 60, 160, 160, 2);
}
else {
    // Desktop browsers
    ig.main('#canvas', MyGame, 60, 240, 160, 2);
}
```

如果您希望提供不同的 HTML 文件，每个设备，你可以用简单的 index.php。此示例假定您有一个索引 iphone.html、 索引 ipad.html 和索引 desktop.html 的文件相同的目录中。

```
<?php
if( preg_match('/iphone|android/i', $_SERVER['HTTP_USER_AGENT']) ) {
        include('index-iphone.html');
}
else if( preg_match('/ipad/i', $_SERVER['HTTP_USER_AGENT']) ) {
        include('index-ipad.html');
}
else {
        include('index-desktop.html');
}
?>
```

始终呈现在本机分辨率

如果您的画布上的一个像素不直接对应一个像素的设备的屏幕在整个画布上已通过浏览器扩展在前所示 – – 这是极为缓慢。

大多数移动浏览器支持的视区元标记。具有此标记，您可以锁定为 1，即没有缩放页面的缩放级别。

```
<meta name="viewport" content="width=device-width;
        initial-scale=1; maximum-scale=1; user-scalable=0;"/>
```

这已经可以确保画布将呈现在其本机分辨率。

当然，有一个例外： 在 iPhone 4 和第 4 位。代 iPod 触摸实际上对页面进行缩放了 2 倍，如果指定缩放 1。他们显示有这么多每英寸点数是通常一个像素大小的一切将缩放两次对这些设备出现在相同的大小。即，他们有的设备像素比 2。

如果我们扩展为两次通过 ig.main() 功能的 iPhone 4 的画布，它将显示它不是我们想要仍将在其原始分辨率不会呈现较大。我们其实现在必须缩减画布的显示大小 （不在画布的内部决议） 再次用 CSS。

例如 160x160px 本机分辨率的游戏，会有 ig.main() 的下列要求：

```
if( ig.ua.iPhone4 ) {
    // The game's native resolution is 160x160. It will be
    // scaled up 4x for the iPhone4, resulting in a drawing
    // resolution of 640x640 px
    ig.main('#canvas', MyGame, 60, 160, 160, 4);
```

```
}
else {
    // For all other devices (including desktop browsers),
    // the game will be scaled up 2x, resulting in a drawing
    // resolution of 320x320 px
    ig.main('#canvas', MyGame, 60, 160, 160, 2);
}
```

和下面的 CSS：

```
#canvas {
        /* Remember, those 320 "px" will be displayed over 640 pixels
        on the iPhone 4's screen, that's why we have to render
        the game at 640x640 pixels. */

        width: 320px;
        height: 320px;
}
```

这是一个好地作用于所有 iOS 设备的完整的 HTML 文件。这是本质上用于 Biolab 灾难的 HTML。它还定义了可以绑定到通过游戏操作的四个按钮。ig.input.bindTouch()。您可以下载的图像文件，用于这些按钮在这里进行。

```
<!DOCTYPE html>
<html>
<head>
        <title>Impact Game</title>
        <style type="text/css">
                html,body {
                        background-color: #000;
                        margin: 0;
                        padding: 0;
                        min-height: 416px;
```

```
                position: relative;
        }

        #canvas {
                width: 320px;
                height: 320px;
        }

        .button {
                background-image: url(media/iphone-buttons.png);
                background-repeat: no-repeat;
                width: 80px;
                height: 96px;
                position: absolute;
                bottom: 0px;

                -webkit-touch-callout: none;
                -webkit-user-select: none;
                -webkit-tap-highlight-color: rgba(0,0,0,0);
                -webkit-text-size-adjust: none;
        }
        #buttonLeft { left: 0; background-position: 0, 0; }
        #buttonRight { left: 80px; background-position: -80px, 0; }
        #buttonShoot { right: 80px; background-position: -160px, 0; }
        #buttonJump { right: 0px; background-position: -240px, 0; }
</style>

<meta name="viewport" content="width=device-width;
        initial-scale=1; maximum-scale=1; user-scalable=0;"/>

<!-- This meta tag ensures that the toolbar at the bottom of the browser
```

```
                is hidden when this page is accessed frome the Home Screen.
-->
        <meta name="apple-mobile-web-app-capable" content="yes" />


        <!-- The icon that should be used when added to the Home Screen -->
        <link rel="apple-touch-icon" href="media/touch-icon.png"/>


        <script type="text/javascript" src="lib/impact/impact.js"></script>
        <script type="text/javascript" src="lib/game/game.js"></script>
</head>


<!-- This will scroll the viewport, so that the navigation bar is no longer visible
-->
<body onload="setTimeout(function(){window.scrollTo(0,0);},1);">
        <div id="game">
                <canvas id="canvas"></canvas>
                <div class="button" id="buttonLeft"></div>
                <div class="button" id="buttonRight"></div>
                <div class="button" id="buttonShoot"></div>
                <div class="button" id="buttonJump"></div>
        </div>
</body>
</html>
```

您还可以使用另一个 meta 标记指定 iPhone 的状态栏的显示方式：

```
<meta name="apple-mobile-web-app-status-bar-style"
content="black-translucent" />
```

尽可能少使用绘图调用

绘制的每个调用具有显著的性能影响。您应该限制您绘制的要求尽可能的少。如果您曾经使用 OpenGL 或 Direct3D 过，这不会对你感到惊讶。

绘制调用基本上是任何绘制任何事的画布 API 的调用。中的影响，这些调用的 api 的大部分都是通过 ig 完成的。图像的.drawtile() 方法。它到处都用：动画使用它绘制当前动画帧和 BackgroundMaps 使用它从其花纹绘制每个平铺的地图。

BackgroundMaps，默认情况下，需要一个抽奖调用每个拼贴所显示的因为它们很容易最大性能猪。

幸运的是，BackgroundMaps 跟 pre-renders 地图成大块的特殊模式。这种方式，引擎只有呈现一个或两个数据块的每个帧，而不是几个几百瓷砖。

您可以在任何时间 （通常在加载一个级别后右） 启用此模式通过.prerender 属性。例如：

```
// in your ig.Game subclass
loadLevel: function( level ) {
    this.parent( level );

    // Enable the pre-rendered background mode for all
    // mobile devices
    if( ig.ua.mobile ) {
        for( var i = 0; i < this.backgroundMaps.length; i++ ) {
            this.backgroundMaps[i].preRender = true;
        }
    }
}
```

请注意，如果您使用此模式，所有的背景动画都会丢失。

如果您使用的粒子对游戏没有什么作用（只是看起来很漂亮），您应该在手机平台上限制它们的数量。

比如 Biolab Disaster 中的敌人 Blob 的 .kill() 方法是这样做的：

```
kill: function() {
    // Limit number of gibs on mobile platforms
    var gibs = ig.ua.mobile ? 5 : 30;

    for( var i = 0; i < gibs; i++ ) {
        ig.game.spawnEntity( EntityBlobGib, this.pos.x, this.pos.y );
    }
    this.parent();
},
```

声音

截至目前，所有手机平台都在努力发挥 HTML5 的音频功能。在 iOS 中，一次可以播放一个单一的声音，但仍然有问题，就是目前还不能预加载声音文件。

最好的办法是完全禁用这些平台上的声音。这也是有好处的，就是你根本不需要加载声音文件（文件常常很大）了。

您可以用 ig.Sound.enabled 属性全局禁用声音。

```
//对手机设备禁用所有声音
if( ig.ua.mobile ) {
    ig.Sound.enabled = false;
}

// Start the game
ig.main(…)
```

调试

影响、 于 1.18 版本开始，features 一些先进的设施，可帮助您看到的调试什么每时每刻你游戏。

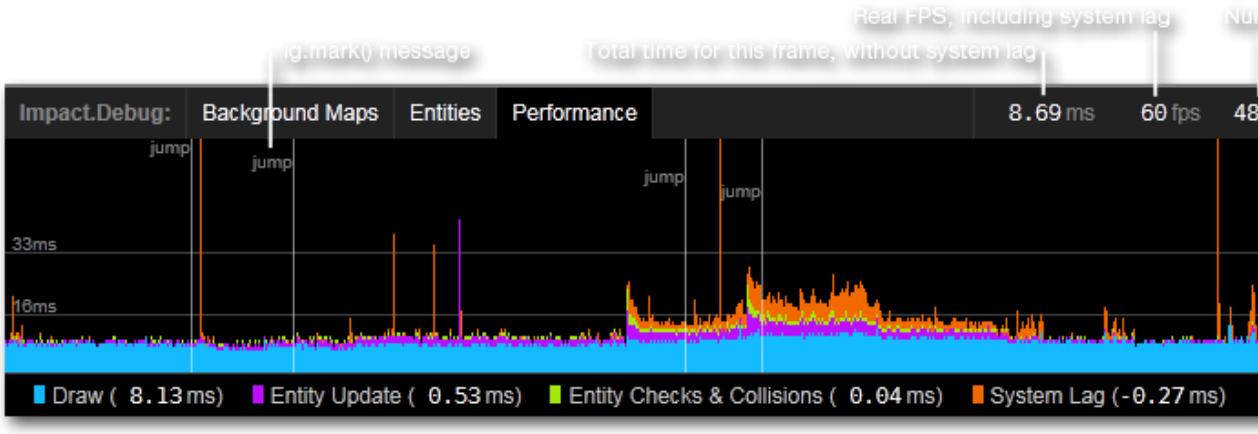
通过只要求在您的代码中的 impact.debug.debug 模块，可以启用调试菜单和其职能。例如，在您的 main.js 文件：

```
ig.module(
    'game.main'
)
.requires(
    'impact.game',
    'impact.font',
    'impact.debug.debug' // <- Add this
)
.defines(function(){

…

});
```

概述

当调试模块加载时，调试菜单将显示在游戏中的页的底部。默认情况下，此菜单 3 板 （背景图、 实体和性能），但您可以添加您选择一些游戏特定调试选项面板。

性能面板可能是最有用的一个，但还需要解释的一点：



两个水平线，标记为 16ms 年和 33ms，说明了两个常见的帧速率为游戏。如果您管理 16ms 年马克下呆下去，你的游戏将以每秒 60 帧运行 （缺省值为 1000ms / 60 帧/秒 = 16.7ms） ；根据 33ms 年它仍然达到每秒 30 帧。

您还可以看到此屏幕抓图，从屏幕的实际绘图无疑是最大的性能猪。启用的呈现模式，为背景地图大大提高这 （您还可以启用这从背景地图面板）。

系统滞后度量也是很有趣的。这是本质上的浏览器是晚开始处理下一帧的时间。例如当当前帧时完全结束 10ms 年后，浏览器应该安排下一帧中 6 毫秒，为了达到 60 fps。

但是 ,某些浏览器的 JavaScript 引擎需要很长的时间来释放未使用的内存（"垃圾回收"），有时高达 100 毫秒的游戏或更多的完全停止。可悲的是，没有什么，作为游戏的开发者，你可以对此-以外的其他浏览器供应商抱怨。

调试消息

Ig 核心有 3 调试消息功能。不过，这些函数将只做一些调试模块加载时。当您想要放你的游戏-只需删除调试模块和所有调试消息都已经过去了 ,这很有用。

```
ig.log(…)
```

浏览器的 console.log() 的别名。它是安全的即使浏览器没有控制台定义的对象，像 IE9 的开发人员工具不使用此函数。

```
ig.show( name, number )
```

浏览器的 console.log() 的别名。它是安全的即使浏览器没有控制台定义的对象，像 IE9 的开发人员工具不使用此函数。

```
// assuming #player# is an entity
ig.show( 'x vel', player.vel.x.round() ); // round before output
```

```
ig.mark( msg, [color] )
```

在当前时间性能图添加味精。颜色可选参数接受诸如 '# f0f' 的 CSS 颜色字符串。如果性能面板是目前处于非活动状态，该消息将被忽略。

添加您自己的面板

要有更多的游戏特定调试选项菜单，您可以添加您自己的面板。例如，如果您希望能够快速启用和禁用 CollisionMaps 的碰撞检测，执行此操作：

```
ig.module(
    'game.my-collision-debug-panel'
)
.requires(
    'impact.debug.menu',
    'impact.collision-map'
)
.defines(function(){
```

```
// Overwrite the CollisionMap's trace method, to check for a custom flag
ig.CollisionMap.inject({
    trace: function( x, y, vx, vy, objectWidth, objectHeight ) {
        if( ig.CollisionMap._enabled ) {
            // Just call the original trace method
            return this.parent( x, y, vx, vy, objectWidth, objectHeight );
        }
        else {
            // Return a dummy trace result, indicating that the object
            // did not collide
            return {
                collision: {x: false, y: false},
                pos: {x: x+vx, y: y+vy},
                tile: {x: 0, y: 0}
            };
        }
    }
});

// This is a 'static' property of ig.CollisionMap. It's not per instance.
ig.CollisionMap._enabled = true;



// Add a panel to the debug menu that allows us to toggle the _enabled flag
// for ig.CollisionMap
ig.debug.addPanel({
    type: ig.DebugPanel,
    name: 'collisionMap',
    label: 'Collision Map',
```

```
    // Toggle switches for this panel
    options: [
        {
            name: 'Enable Collisions',

            // When the toggle switch is clicked, it will flip the property
            // value for the given object.
            // In this case 'ig.CollisionMap._enabled'
            object: ig.CollisionMap,
            property: '_enabled'
        }
    ]
});

});
```

这将创建新的免疫球蛋白。DebugPanel 与切换切换到启用的碰撞，并将其添加到调试菜单。

如果您需要更多只是一些简单的切换开关，还可以子类 ig 从您自己面板类。DebugPanel。例如：

```
ig.module(
    'game.my-fancy-debug-panel'
)
.requires(
    'impact.debug.menu',
    'impact.entity',
    'impact.game'
)
.defines(function(){
```

```
// Overwrite the Game's loadLevel() method to call a custom method
// on our panel, after the level is loaded
ig.Game.inject({
    loadLevel: function( data ) {
        this.parent(data);

        // 'fancypanel' is the name we give this panel in the
        // call to ig.debug.addPanel()
        ig.debug.panels.fancypanel.load(this);
    }
});

// Overwrite the Entity's update() method, so we can disable updating
// for a particular entity at a time
ig.Entity.inject({
    _shouldUpdate: true,
    update: function() {
        if( this._shouldUpdate ) {
            this.parent();
        }
    }
});

MyFancyDebugPanel = ig.DebugPanel.extend({

    init: function( name, label ) {
        // This creates the DIV container for this panel
        this.parent( name, label );

        // You may want to load and use jQuery here, instead of
        // dealing with the DOM directly...
```

```
        this.container.innerHTML =
            '<em>Entities not loaded yet.</em>';
    },


    load: function( game ) {
        // This function is called through the loadLevel() method
        // we injected into ig.Game


        // Clear this panel
        this.container.innerHTML = '';


        // Find all named entities and add an option to disable
        // the movement and animation update for it
        for( var i = 0; i < game.entities.length; i++ ) {
            var ent = game.entities[i];
            if( ent.name ) {
                var opt = new ig.DebugOption( 'Entity ' + ent.name, ent,
'_shouldUpdate' );
                this.addOption( opt );
            }
        }


    },


    ready: function() {
        // This function is automatically called when a new Game is created.
        // ig.game is valid here!
    },


    beforeRun: function() {
        // This function is automatically called BEFORE each frame
        // is processed.
```

```
    },

    afterRun: function() {
        // This function is automatically called AFTER each frame
        // is processed.
    }
});

ig.debug.addPanel({
    type: MyFancyDebugPanel,
    name: 'fancypanel',
    label: 'Fancy Panel'
});

});
```

烘焙

现在你已经完成了你的游戏，想要在因特网上展示一下吗？太棒了！然而，在这之前，你应该将所有的源文件打包成一个大文件。这将为你的访问者大大缩短初始加载时间。

Impact 自带一个为你打包的 PHP 脚本。你可以从命令行中调用这个脚本，或者在 Windows 中双击 tools 目录下的 bake.bat 。 在 MacOSX 中打开一个终端窗口，进入 tools/ 目录，写 ./bake.sh 。

如果有错误信息，请确保所有的路径都正确。你可以用文本编辑器打开 bake.bat 或 bake.sh 。你需要修改的仅有的两行是：

```
SET GAME=lib/game/main.js


SET OUTPUT_FILE=game.min.js
```

游戏变量应指向你的游戏主要.js 文件 （您加载 <script> 标记的.html 页面中的一个）。

OUTPUT_FILE 确定将在其中写入烤的脚本文件。

在 Windows 上也有确保 bake.bat 可以找到 php.exe，您的系统上。您可以添加到您的 PATH 环境变量的 PHP 的安装路径 （询问 google） 或编辑直接指向您 php.exe bake.bat。

如果没有错误，脚本完成后，您可以找到 game.min.js （或无论您将它设置为在 bake.bat） 影响目录中。现在，您可以在您的.html 的你以前过的两个文件而不是加载此.js 文件。

如对此进行更改

```
<script type="text/javascript" src="lib/impact/impact.js"></script>
<script type="text/javascript" src="lib/game/main.js"></script>
```

对此

```
<script type="text/javascript" src="game.min.js"></script>
```

Game.min.js 然后是你必须上传或发布的唯一的源文件。

删除源代码

这是删除演示游戏的源代码。

你购买了一个许可证后，您可以在您个人的下载页上，下载完整的包，所有的图像、 声音、.html 文件和影响游戏引擎。

一个有趣的游戏是游戏加载时，就从源图像生成的液晶屏外观。例如的弹跳球雪碧是刚刚 4 x 4 像素的大小。

```
ig.module(
    'drop'
)
.requires(
    'impact.game',
    'impact.entity',
    'impact.collision-map',
    'impact.background-map',
    'impact.font'
)
.defines(function(){




// The Backdrop image for the game, subclassed from ig.Image
// because it needs to be drawn in it's natural, unscaled size,
FullsizeBackdrop = ig.Image.extend({
    resize: function(){ /* Do Nothing */ },
    draw: function() {
        if( !this.loaded ) { return; }
        ig.system.context.drawImage( this.data, 0, 0 );
    }
});
```

```
// The Collectable Coin Entity
EntityCoin = ig.Entity.extend({
    size: {x:6, y:6},
    offset: {x:-1, y:-1},
    animSheet: new ig.AnimationSheet( 'media/coin.png', 4, 4 ),
    type: ig.Entity.TYPE.B,

    sound: new ig.Sound('media/coin.ogg'),

    init: function( x, y, settings ) {
        this.addAnim( 'idle', 0.1, [0,1] );
        this.parent( x, y, settings );
    },

    update: function() {
        this.parent();
        if( this.pos.y - ig.game.screen.y < -32 ) {
            this.kill();
        }
    },

    pickup: function() {
        ig.game.score += 500;
        this.sound.play();
        this.kill();
    }
});
```

```
// The Bouncing Player Ball thing
EntityPlayer = ig.Entity.extend({
    size: {x:4, y:4},
    checkAgainst: ig.Entity.TYPE.B,

    animSheet: new ig.AnimationSheet( 'media/player.png', 4, 4 ),

    maxVel: {x: 50, y: 300},
    friction: {x: 600, y:0},
    speed: 300,
    bounciness: 0.5,
    sound: new ig.Sound('media/bounce.ogg'),

    init: function( x, y, settings ) {
        this.addAnim( 'idle', 0.1, [0] );
        this.parent( x, y, settings );
    },

    update: function() {
        // User Input
        if( ig.input.state('left') ) {
            this.accel.x = -this.speed;
        }
        else if( ig.input.state('right') ) {
            this.accel.x = this.speed;
        }
        else {
            this.accel.x = 0;
        }

        this.parent();
```

```
    },

    handleMovementTrace: function( res ) {
        if( res.collision.y && this.vel.y > 32 ) {
            this.sound.play();
        }
        this.parent(res);
    },

    check: function( other ) {
        // The 'other' entity must be a coin, because we
        // only have two entity types (coins and the player)
        other.pickup();
    }
});



// A Custom Loader for the game, that, after all images have been
// loaded, goes through them and "pixifies" them to create the LCD
// effect.
DropLoader = ig.Loader.extend({
    end: function() {
        for( i in ig.Image.cache ) {
            var img = ig.Image.cache[i];
            if( !(img instanceof FullsizeBackdrop) ) {
                this.pixify( img, ig.system.scale );
            }
        }
        this.parent();
    },
```

```
    // This essentially deletes the last row and collumn of pixels for
    // each upscaled pixel.
    pixify: function( img, s ) {
        var ctx = img.data.getContext('2d');
        var px = ctx.getImageData(0, 0, img.data.width, img.data.height);

        for( var y = 0; y < img.data.height; y++ ) {
            for( var x = 0; x < img.data.width; x++ ) {
                var index = (y * img.data.width + x) * 4;
                var alpha = (x % s == 0 || y % s == 0) ? 0 : 0.9;
                px.data[index + 3] = px.data[index + 3] * alpha;
            }
        }
        ctx.putImageData( px, 0, 0 );
    }
});



// The actual Game Source
DropGame = ig.Game.extend({
    clearColor: '#c7e300',
    gravity: 240,
    player: null,

    map: [],
    score: 0,
    speed: 1,
```

```
    tiles: new ig.Image( 'media/tiles.png' ),
    backdrop: new FullsizeBackdrop( 'media/backdrop.png' ),
    font: new ig.Font( 'media/04b03.font.png' ),
    gameOverSound: new ig.Sound( 'media/gameover.ogg' ),

    init: function() {
        ig.system.smoothPositioning = false;

        ig.input.bind(ig.KEY.LEFT_ARROW, 'left');
        ig.input.bind(ig.KEY.RIGHT_ARROW, 'right');
        ig.input.bind(ig.KEY.ENTER, 'ok');

        // The first part of the map is always the same
        this.map = [
            [0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0],
            [0,0,0,1,1,0,0,0],
            [0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0],
        ];

        // Now randomly generate the remaining rows
        for( var y = 8; y < 18; y++ ) {
            this.map[y] = this.getRow();
        }

        // The map is used as CollisionMap AND BackgroundMap
        this.collisionMap = new ig.CollisionMap( 8, this.map );
```

```
        var bgmap = new ig.BackgroundMap( 8, this.map, this.tiles );

        // Add the bgmap to the Game's array of BackgroundMaps
        // so it will be automatically drawn by .draw()
        this.backgroundMaps.push( bgmap );

        this.player = this.spawnEntity( EntityPlayer, 30, 16 );
    },


    getRow: function() {
        // Randomly generate a row of blocks for the map. This is a naive
        // approach, that sometimes leaves the player hanging with no
        // block to jump to. It's random after all.
        var row = [];
        for( var x = 0; x < 8; x++ ) {
            row[x] = Math.random() > 0.93 ? 1 : 0;
        }
        return row;
    },


    placeCoin: function() {
        // Randomly find a free spot for the coin, max 12 tries
        for( var i = 0; i < 12; i++ ) {
            var tile = (Math.random() * 8).ceil();
            if(
                this.map[this.map.length-1][tile] &&
                !this.map[this.map.length-2][tile]
            ) {
                var y = (this.map.length-1) * 8;
```

```
                var x = tile * 8 + 1;
                this.spawnEntity( EntityCoin, x, y );
                return;
            }
        }
    },


    update: function() {
        if( ig.input.pressed('ok') ) {
            ig.system.setGame( DropGame );
        }

        if( this.gameOver ) {
            return;
        }

        this.speed += ig.system.tick * (10/this.speed);
        this.screen.y += ig.system.tick * this.speed;
        this.score += ig.system.tick * this.speed;

        // Do we need a new row?
        if( this.screen.y > 40 ) {

            // Move screen and entities one tile up
            this.screen.y -= 8;
            for( var i =0; i < this.entities.length; i++ ) {
                this.entities[i].pos.y -= 8;
            }

            // Delete first row, insert new
```

```
            this.map.shift();
            this.map.push(this.getRow());

            // Place coin?
            if( Math.random() > 0.5 ) {
                this.placeCoin();
            }
        }
        this.parent();

        // check for gameover
        var pp = this.player.pos.y - this.screen.y;
        if( pp > ig.system.height + 8 || pp < -32 ) {
            this.gameOver = true;
            this.gameOverSound.play();

        }
    },


    draw: function() {
        this.backdrop.draw();

        if( this.gameOver ) {
            this.font.draw( 'Game Over!', 32, 32, ig.Font.ALIGN.CENTER );
            this.font.draw( 'Press Enter', 32, 48, ig.Font.ALIGN.CENTER );
            this.font.draw( 'to Restart', 32, 56, ig.Font.ALIGN.CENTER );
        }
        else {
            for( var i = 0; i < this.backgroundMaps.length; i++ ) {
                this.backgroundMaps[i].draw();
```

```
            }

            for( var i = 0; i < this.entities.length; i++ ) {
                this.entities[i].draw();
            }
        }

        var s = this.score.floor().toString();
        this.font.draw( s, 62, 2, ig.Font.ALIGN.RIGHT );
    }
});

// Start the game - 30fps, 64x96 pixels, scaled up 5x
ig.main('#canvas', DropGame, 30, 64, 96, 5, DropLoader );

});
```

Box2D 物理引擎

简介

影响提供了一些基本的游戏实体的碰撞检测和响应。这往往是足够的 Jump'n'Run 或 RPG 游戏，但有时你需要一些更准确和真实的物理。这是其中**的 Box2D** 的用武之地。Box2D 的是一个物理引擎，最初写的 C ++ **移植到 ActionScript** 和使用数百种游戏，包括非常流行的**愤怒 Brids**。

最近，ActionScript 版本的 Box2D 的已**转换为 JavaScript** 乔纳斯·瓦格纳。因此，让我们的影响，在使用！

我做了一些快速搜索/替换乔纳斯·瓦格纳的源把自身的 Box2D 的变化*命名空间*。

所有的Box2D类是现在的成员B2。对象。所以代替b2AABB ,b2Vec或b2World 我们现在可以参考 b2.AABB，b2.Vec，b2.World 等。我也裹成一个模块的源代码，很容易影响可装载。

请参阅 **Box2D 的 2.0.2 手册**。

演示

传送插座物理演示

此演示的源代码，可以发现您的**下载页面**上。

Box2D 的基本用法

我们需要做的第一件事是成立 Box2D 的世界。对于这一点，我们只需要定义一个边界框和重力矢量和创建一个实例 b2.World：

```
var boundingBox = new b2.AABB();
boundingBox.lowerBound.Set( 0, 0 );
boundingBox.upperBound.Set( 1024, 512);

var gravity = new b2.Vec2( 0, 30 );

// Create the world and make it globally available as part of ig.
ig.world = new b2.World( boundingBox, gravity, true );
```

与世界的准备，我们不能给它添加一些对象。

```
var bodyDef = new b2.BodyDef();
bodyDef.position.Set( 100, 100 );

var body = ig.world.CreateBody( bodyDef );
```

```
var shape = new b2.PolygonDef();
shape.SetAsBox( 5, 5 );
body.CreateShape( shape );
```

请注意，我们设置的位置（100，100）指定对象的*起源*-它的中心点。这是从不同的影响，其中一个实体的位置设置为上层实体的左上角。此外，SetAsBox 方法采用半宽和半高参数。因此，框，我们创建是由大小 10 个单位 10，因为它的中心是在（100,100），（95,95）（105105），它会伸展。

一切设置后，我们可以提前在我们的游戏的更新方法的物理模拟。在 b2.World.Step（ ）方法有两个参数：在几秒钟的时间步长和内部重复计算。ig.system.tick 提供以来通过的最后一帧的时间-这正是时间我们的物理模拟推进：

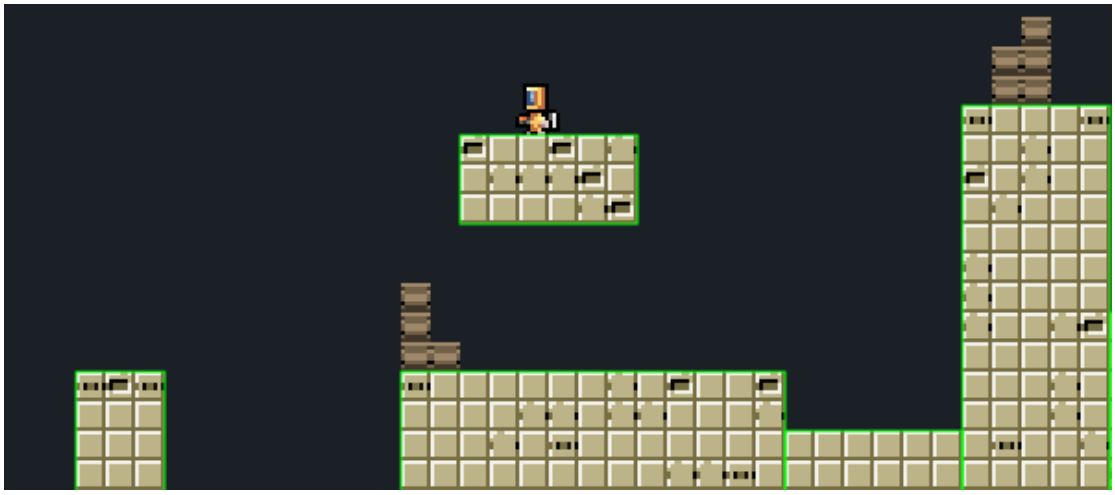
```
update: function() {
    ig.world.Step( ig.system.tick, 5 );
    this.parent();
},
```

设立碰撞地图和实体

在 Box2D 单位调整工作，以及米。你的对象应该是大约 0.1 米和 10 米的大小-这意味着大多数游戏，只是以物理为基础的像素大小，可以引进一些文物之间。因此，我们使用一个全球性的比例因子，转换像素米：b2.SCALE。默认情况下，它被设置为 0.1 -这意味着，1 个像素等于 0.1 米。

现在，整合 Box2D 的影响，这将是很好，有一个基本的实体类，使用 Box2D 的基地游戏类创建的物理世界。对于这一点，我写的的 ig.Box2DGame 和 ig.Box2DEntity 班，为你做所有的脏活。看到物理学例如游戏的源代码。

从碰撞地图。loadLevel（ ）的方法 ig.Box2DGame 创建物理世界。然而，不是每瓦只创建一个物理对象 ,它试图找到较大 的矩形区域 ,瓷砖组合成一个对象。在我的测试，这项工作相当不错，甚至更大的游戏世界。Box2D 的性能似乎将主要由碰撞的数量，而不是届世界中的对象数量的约束。



ig.Box2DGame 有一个特殊的的财产 debugCollisionRects，当设置为 true ，将吸引所有的绿色轮廓的碰撞盒。

自动 ig.Box2DEntity 创建实体的大小与物理身体。其更新（ ）方法需要从物理模拟人体的位置，并将其转换回影响的单位系统。此更新方法完全绕过一个基地 ig.Entity 正常移动实体。

因此，要定义一个简单的物理启用箱实体，所有您需要做的是继承 ig.Box2DEntity：

```
// Subclassing ig.Box2DEntity instead of ig.Entity inherits
// everything needed for the physics simulation
EntityCrate = ig.Box2DEntity.extend({
    size: {x: 8, y: 8},

    // Collision is already handled by Box2D!
    collides: ig.Entity.COLLIDES.NEVER,
```

```
    animSheet: new ig.AnimationSheet( 'media/crate.png', 8, 8 ),

    init: function( x, y, settings ) {
        this.addAnim( 'idle', 1, [0] );
        this.parent( x, y, settings );
    }
});
```

周围移动你的实体，使用 Box2D 中的 ApplyForce 和 ApplyImpulse 功能。例如，我们的球员 实体：

```
update: function() {
    …

    // jetpack
    if( ig.input.state('jump') ) {
        this.body.ApplyForce(new b2.Vec2(0,-30), this.body.GetPosition());
    }

    …

    this.parent();
}
```

类参考

核心和系统

核心 -模块定义，模块加载和的 ig.main（ ）和实用功能

- 类的所有类的基类

系统 -主机当前的游戏对象，并保持运行循环。

装载机 -预载的所有资产（图像和声音），并开始给游戏完成时。

逻辑

游戏 -你的游戏的主要枢纽。它承载的所有当前活动的实体，背景地图和碰撞地图和照顾，更新和借鉴一切。

实体 -在游戏世界的互动对象通常是从这个基地的实体类的子类。它提供动画，绘画和基本物理。

输入 -处理所有键盘和鼠标输入。

定时器 -定时的东西。

地图 -基地类 CollisionMaps 和 BackgroundMaps 的。

CollisionMap -针对 2D 瓷砖地图跟踪对象

图形

图片 -包装周围的图像资源（PNG，GIF 或 JPEG）。它需要加载和缩放源图像。

动画 -动画实体和背景图砖的动画张

AnimationSheet -载入图片动画表，并指定每个动画帧的宽度和高度

字体 -装入一个位图字体图像和绘制文本。

BackgroundMap -绘制一个 2D 的瓷砖地图

听起来

声音 -加载 一个声音文件被用来作为背景音乐或游戏的声音。

音乐 -提供播放背景音乐，并创建一个播放列表的功能。

SoundManager -负载为 ig.Music 和 ig.Sound 的声音文件。

核心

定义模块 impact.impact 的

描述

IG 对象提供模块的定义和负荷能力，以及一些实用功能。应包裹在所有的代码模块，因此它可以正确加载和烘烤。

IG 是不是一个一个 ig.Class 的实例，但只是一个简单的 JavaScript 对象，因此不能被继承。

模块定义

```
ig.module(
```

```
    'game.my-file'
)
.requires(
    'impact.game',
    'impact.image',
    'game.other-file'
)
.defines(function(){

    / /这个模块的代码

}) ;
```

定义新的模块名称，要求所有指定的模块。当加载所有所需的模块，传递给.defines() 函数的执行。

如果该模块不需要任何其他模块，对.requires() 的调用可以省略，如在

```
ig.module(
    'name'
)
.defines(function(){

    // code

});
```

模块的名称，直接对应于其文件的位置和名称。例如文件 lib/mygame/my-file.js 必须定义一个模块名称 'mygame.my-文件'，因此它可以正确加载从其他模块。

您可以将多个模块放入一个文件，但请注意，从外部文件引用的模块必须有它自

己的文件。

属性

ig.baked

由 bake.php 设置为 true，跳过“未解决的依赖”检查时加载模块。默认为 false。看到烘烤。

ig.game

的当前运行 ig.Game 的实例，由 ig.main（ ）或 system.setGame（ ）创建。如果没有比赛已经开始，ig.game 是空的。

ig.global

全球范围内的对象（如窗口同时在浏览器中运行时）

ig.nocache

默认情况下，ig.nocache 是一个空字符串。当网站加载与 game.html？的 NOCACHE 或 ig.setNocache（真）被称为中，ig.nocache 是设置一个随机的参数（如“23478976”）上的网址添加到绕过浏览器的缓存。

这是所有影响的装载功能：加载模块，图像和声音。

ig.version

的在 X.XX 形式的影响游戏引擎的版本字符串

ig.ua

布尔值，表示如果用户代理是以下设备之一：

ig.ua.iPhone - 适用于所有 iPhone 和 iPod Touch 设备

ig.ua.iPhone4 - 真正的 iPhone 4 和第 4 代 iPod Touch 设备

ig.ua.iPad - 真正的 iPad 设备

ig.ua.iOS - 适用于所有 iPhone，iPod Touch 或 iPad 设备

ig.ua.android - 真正的 Android 设备

ig.ua.mobile - 真正的所有 iPhone，iPad 或 Android 设备

新的 1.15

ig.ua.pixelRatio - CSS 像素来硬屏像素的比例。一般的台式电脑，2 个为 iPhone 4 或类似 1。

ig.ua.screen.width，ig.ua.screen.height - 设备的硬件像素的屏幕大小。

ig.ua.viewport.width，ig.ua.viewport.height - CSS 像素在当前视口的大小。

构造方法

```
ig.main( canvas, gameClass, fps, width, height, [scale], [loaderC
lass] );
```

初始化 ig.system ，ig.input ，ig.soundManager 和 ig.music 并启动预载。当 Preloader 完成后，一个 gameClass 实例在 ig.game 创建和运行的循环开始。

画布 canvas 元素，在该游戏将绘制字符串 ID 选择。例如：“ # myCanvas

其中一个实例将被创建。run（ ）方法被称为每秒 FPS 倍的 gameClass 的 ig.Game 子类。

FPS 期望帧每秒

宽度，高度的游戏画面尺寸

规模缩放的游戏画面。例如宽度为 320 和 2 分 ,将导致在一个 640 像素宽的 canvas 元素。默认值是 1（无缩放）。

loaderClass 一个子类的 ig.Loader 作为 preloader 的。默认是 ig.Loader 的类本身。

请注意，FPS 参数表示每秒所需的帧。如果 FPS 下降低于 20，步长将被限制在第二个二十分之一，从根本上减缓比赛下来。例如可以呈现时，只有 10 帧，“在比赛第二”的两个“实时秒”。看到 ig.Timer.maxStep。

调试功能

1.18 中的新功能

定义了三个不同的功能 ,用于调试输出的影响： ig.log()、 ig.show() 和 ig.mark()。然而 ,这些函数实际上只做些调试模块加载时。有关详细信息 ,请参阅调试消息。

总是保存时要调用这些函数，即使没有加载调试模块。

实用程序函数

```
ig.$( selector )
```

获取 DOM-从给定选择程序依法页面元素。目前支持唯一的元素的名称和 Id。

当的元素名称是指定 （例如 ' 头'）返回匹配的元素的数组。当指定 ID 的选择器 （例如 '#canvas'）作为将返回具有该 ID 的单个元素。

（此行为有些矛盾和更改可能在将来的版本）

```
ig.$new( name )
```

用给定的名称创建一个新的 DOM 元素。例如：

```
var audio = ig.$new('Audio');
```

```
ig.copy( object )
```

返回给定对象的深层副本。

请注意，此函数不能复制 `ig.Class` 或子类和只是它们返回 uncopied。

```
ig.ksort( obj )
```

采用对象并返回一个数组，按的键的对象的属性进行排序。例如：

```
var unsorted = { z: 'baz', a: 'bar', b: 'foo' };
var sorted = ig.ksort( unsorted );

sorted[0]; // => bar
sorted[1]; // => foo
sorted[2]; // => baz
```

```
ig.merge( target, extended )
```

将扩展的对象以递归方式的所有属性都合并到目标对象 ,覆盖现有的属性。例如：

```
var settings = { width: 256, height: 128 };
var extendedSettings = { width: 512, foo: 'bar' };

ig.merge( settings, extendedSettings );

settings.width; // => 512
settings.height; // => 128
settings.foo; // => bar
```

```
ig.setNocache( set )
```

当设置为 true 时 ,则 ig.nocache 将被设置为随机参数的字符串。时设置为 false，则 ig.nocache 将设置为空字符串。

原生 JavaScript 对象扩展

影响扩展了一些 Javascript 本机对象如编号、 数组和函数。它不会扩展对象，以便您可以安全地仍使用百循环。

```
Number.map( fromMin, fromMax, toMin, toMax )
```

映射到另一个范围值的数字范围。数字不是夹紧。

```
var num = 50;
num.map( 0, 100, 200, 400 ); // => 300

(50).map( 0, 100, 200, 400 ); // => 300
(25).map( 0, 100, 200, 400 ); // => 250
```

```
(150).map( 0, 100, 200, 400 ); // => 500
```

```
Number.limit( min, max )
```

夹紧范围的最小和最大数目。

```
(75).limit( 50, 100 ); // => 75

(25).limit( 50, 100 ); // => 50

(150).limit( 50, 100 ); // => 100
```

```
Number.round( [precision] )
```

轮到精密十进制数字的地方 （默认为 0）。

```
(3.1415).round(); // => 3

(3.1415).round(2); // => 3.14
```

```
Number.floor()
```

Math.floor() 方便方法

```
Number.ceil()
```

Math.ceil() 方便方法

```
Number.toInt()
```

放弃小数的数字。对于负数，行为是不同的 Number.floor()。

```
(2.8).toInt(); // => 2

(-2.8).toInt(); // => -2


(2.8).floor(); // => 2
```

```
(-2.8).floor(); // => -3
```

```
Number.toDeg()
```

1.19 中的新功能

将弧度转换为度

```
(0.7853981633974483).toDeg(); // => 45
```

```
Number.toRad()
```

1.19 中的新功能

度转换为弧度

```
(45).toRad(); // => 0.7853981633974483
```

```
Array.erase( item )
```

查找和删除给定的数组中的项。这种方法操作中的地方的数组。

```
var a = [4,8,13,15,16,23,42];
a.erase(13);
a; // => [4,8,15,16,23,42]
```

```
Array.random()
```

返回一个随机的元素的数组。

```
Function.bind( bind )
```

设置此绑定函数内部。Function.bind() 是有助于维护此回调方法的对象。

```
var foo = {
    bar: 100,
    inc: function() {
        this.bar++;
        console.log( this.bar );
    }
};

// Without using bind(), the "this" inside foo.inc() would refer to the
// "window" instead to itself, when called through setInterval()
setInterval( foo.inc.bind(foo), 1000 );
```

类

定义模块 impact.impact 的

简介

```
// Create a new class "Person"
var Person = ig.Class.extend({
    name: '',
    init: function( name ) {
        this.name = name;
    }
});

// Create another class by extending the "Person" class
var Ninja = Person.extend({
    init: function( name ) {
        this.parent( 'Ninja: ' + name );
```

```
    }
});

// Instatinate an object of the first class
var e = new Person('Generic Person');
e.name; // => Generic Person

// Instatinate an object of the second class
var p = new Ninja('John Resig');
p.name; // => Ninja: John Resig
```

描述

影响的类对象是基于 John Resig 的简单的 Java 脚本继承代码 ,但它延伸与深复制属性和静态实例。

影响所有的类都是从这个基类派生，你可以用它来创建自己的类。

用法

```
.extend( classDefinition )
```

`.extend()`需要一个 JavaScript 对象 ,并指定新的类的属性和方法。里面的方法，this 指这个类的实例。

```
var Foo = ig.Class.extend({
    bar: 'baz',
    setBar: function( bar ) {
        this.bar = bar;
```

```
    }
});

var foo1 = new Foo();
foo1.setBar( 'bar1' );
foo1.bar; // => bar1

var foo2 = new Foo();
foo2.setBar( 'bar2' );
foo2.bar; // => bar2
```

创建的所有类`.extend()`也将`.extend()`函数，可用于进一步的子类。

```
.inject( classDefinition )
```

新的 1.17

`.inject()`的工作方式类似于`.extend()`但不创建一个新的类-相反，它改变了类到位。如果你想改变影响类的行为，无需改变发动机的源代码，例如插件，这是非常有用的。

```
// Overwrite ig.Image's .resize method to provide your
// own scaling algorithm
ig.Image.inject({
    resize: function( scale ) {
        if( scale == 2 ) {
            this.data = awesome2XScalingAlgorithm( this.data );
        }
        else {
            // Call ig.Image's resize function if scale is not 2
```

```
            this.parent( scale );
        }
    }
});

// The new resize method will also be used in subclasses of
// ig.Image (e.g. ig.Font)
```

```
init: function() {}
```

这个类的`.init()`方法，如果存在的话，一个新的实例被创建时被调用。

```
var InitTest = ig.Class.extend({
    init: function( fparam ) {
        console.log( 'Init called with ' + fparam );
    }
});

var t1 = new InitTest( 'ZOMG' ); // => Init called with ZOMG
```

```
staticInstantiate: function() {}
```

如果一个类有一个 staticInstantiate 功能，它被称为前创建了这个类的新实例。当 staticInstantiate 函数返回空，一个新的类的实例被创建并返回。如果 staticInstantiate 返回非空，则返回它的返回值。

这例如可以用来创建一个 Singleton 类-一类只允许一个实例。

```
var MySingleton= ig.Class.extend({
```

```
    foo: 'bar',
    staticInstantiate: function( ignoredFoo ) {
        if( MySingleton.instance == null ) {
            return null;
        }
        else {
            return MySingleton.instance;
        }
    },

    init: function( foo ) {
        this.foo = foo;
        MySingleton.instance = this;
    }
});

MySingleton.instance = null;

var s1 = new MySingleton( 'baz' );
var s2 = new MySingleton( 'ignored' );

s1.foo; // baz
s2.foo; // baz

s1 == s2; // true
```

请注意，MySingleton.instance 不被定义为一个实例属性，作为一个阶级的财产，但。你还可以将您的类的“静态”的功能，以这种方式。这是类似的静态 Java 或 PHP 中的关键字。

```
this.parent(…)
```

this.parent 里面的方法，始终是指方法与超类的名称相同-如果存在。即覆盖了一个超类的方法时，你仍然可以在子类的方法调用它 this.parent（ ） 。

系统

模块中的定义 impact.system，从 ig.Class 继承

描述

ig.System 启动和停止运行循环，并呼吁对当前游戏对象的 run（ ）方法。同时，它也为家政 ig.Input，并提供了一些实用的方法。

会自动创建一个实例 ig.System 由 ig.main（ ）函数在 ig.system（小写 ）。

构造方法

```
new ig.System( canvasId, fps, width, height, scale )
```

- canvasId 选择一个 ID 字符串，指定用于绘图的画布元素。例如' # 画布
- FPS 这个游戏所需的帧每秒
- 宽度的游戏画面（未缩放）宽度
- 高度的游戏画面（未缩放）高度

- 游戏画面缩放比例因子

构造 ig.System 通常被称为通过 ig.main（ ）和实例提供 ig.system（小写 ）。

属性

```
.canvas
```

canvas 元素。

```
.context
```

'2 D'绘制背景的画布元素。

```
.fps
```

游戏将运行所需的帧每秒。这通常是通过构造由 ig.main（ ） 。如果你改变这个属性，你需要重新启动与运行循环。startRunLoop（ ）变更生效。

```
.realWidth, .realHeight
```

规模大小的游戏画面。例如一个宽度 320 像素和一个。规模的 2 会导致一个。realWidth 的 640。

```
.running
```

真正的，如果一个游戏正在运行，否则返回 false。

```
.scale
```

游戏画面的缩放因子。

```
.smoothPositioning
```

是否使用单像素定位的游戏画面，即使已扩大。

例如，如果规模是 4 。smoothPositioning 是真正的，移动的实体将它移动一个像素一次。smoothPositioning 如果。是假的，最小的运动实体的第一步将是 4 个像素。

虽然是有点“非正宗的” 。smoothPositioning，它提供了实体层和背景层的动作更加流畅的游戏画面缩放时。

默认的是真实的。

```
.tick
```

在几秒钟内，因为最后一帧的时间。移动计算和使用。例如：

```
pos.x = pos.x + vel.x * ig.system.tick;
```

请注意，这个值并不代表实时 ，但 比赛时间。例如，如果浏览器是无法运行在 20 FPS 游戏，整场比赛将放缓。蜱将低于以来的最后一帧的实时。看到 ig.Timer 更详细的解释。

```
.width, .height
```

（未缩放）的游戏画面在像素大小，构造。

方法

```
.clear( color )
```

清除指定的颜色，在游戏画面的颜色是一个 CSS 字符串。如 # f0f “ 。

绘制的 ig.Game's `.draw()`方法调用 `.clear()`与游戏的每一帧`.clearColor`。

```
.getDrawPos( p )
```

到画布上绘图，得到真正的目标位置。这需要的`.scale`和`.smoothPositioning`考虑。

你只需要这种方法 如果你的工作`.context` ,而不是绘画 ,通过 ig.Image ,in.Front 或 in.Background 地图直接。

```
/ /直接使用画布背景下，我们需要采取的
/ / ig.system.scale 考虑。使用 getDrawPos （）
/ /给我们的“规模位置”
ig.system.context.fillText(
    'Test Text',
    ig.system.getDrawPos( x ),
    ig.system.getDrawPos( y )
);
```

```
.resize( width, height, [scale] )
```

新的 1.17

设置(未缩放)像素的画布元素的宽度和高度。如果可选的规模参数是不存在的，规模的因素没有改变。

```
.setGame( gameClass )
```

运行指定 gameClass。停止当前正在运行的游戏（如有），创建一个新的的实例 gameClass 和再次开始运行循环。

这种方法是安全的呼吁，在游戏中的任何地方，作为新的游戏才会开始后，目前的框架已经完成更新和绘图。

装载

模块中的定义 **impact.loader**，从 ig.Class 继承

简介

```
//子类的默认
MyLoader = ig.Loader.extend({


    draw: function() {
        //这里加入你的绘图代码
        // /这一个清除屏幕并提请
        //加载百分比
        var w = ig.system.realWidth;
        var h = ig.system.realHeight;
```

```
        ig.system.context.fillStyle = '#000000';
        ig.system.context.fillRect( 0, 0, w, h );

        var percentage = (this.status * 100).round() + '%';
        ig.system.context.fillStyle = '#ffffff';
        ig.system.context.fillText( percentage, w/2,  h/2 );
    }
});

//调用 ig.main（）为您自定义的类装载
ig.main('#canvas', MyGame, 60, 320, 240, 2, MyLoader);
```

描述

ig.Loader 是默认 preloader 的游戏所需要的所有图像和声音。它显示了一个黑色的背景上的白色的进度条。

你子类 ig.Loader 能以提供自己的绘图代码，或者做一些额外的处理。

请注意，而帆布和其绘图上下文是当 Preloader 执行，你可以不使用任何 ig.Image 或 ig.Sound 资源尚未。

构造

```
new ig.Loader( gameClass, resources )
```

- gameClass 的 ig.Game 类开始，当所有的资源已载入
- 资源资源的阵列，其中每个必须提供。负载（ ）方法接受，如回调，ig.Image 和 ig.Sound。

请注意，一个 Loader 类的一个实例通常是由创建 ig.main() 。

属性

```
.done
```

真正当一切都已经加载，否则假。

```
.gameClass
```

开始 ig.Game()类，交给本实例通过构造。

```
.resources[]
```

通过构造阵列的资源加载，交给此实例。

```
.status
```

加载资源的分数。例如，如果已加载资源 10 20，状态是 0.5。这只能算作资源的数量，而不是它们的大小。

方法

```
.draw()
```

所谓的 60 倍，在负载每秒更新进度条。

```
.end()
```

当所有的资源都完成加载调用。这通常要求 ig.system.setGame（ ）装载机。gameClass。

```
.load()
```

启动加载的所有资源。通常称为由 ig.main（ ）。

```
.loadResource( res
```

启动加载从资源之一。资源阵列。，RES 必须是一个对象，有一个实例。负载（ ）方法，如 ig.Image 和 ig.Sound。

游戏

在模块中定义 impact.game，从 ig.Class 继承

简介

```
//创建自己的游戏类
MyGame = ig.Game.extend({

    init: function() {

        //初始化你的游戏世界，结合一些/按键等
    ig.input.bind( ig.KEY.SPACE, 'jump' );

        this.loadLevel( LevelMyGame1 );

    }

});
```

```
// 开始你的游戏
// 60fps 的，你的游戏的 320x240 像素，缩小了 2 倍
ig.main('#canvas', MyGame, 60, 320, 240, 2 );
```

描述

ig.Game 是你的游戏的主要枢纽。它承载了所有当前活动的实体，背景地图和碰撞地图。你可以子类从你自己的游戏类 ig.Game。

其.update()和.draw()方法碰撞检测，检查对彼此的实体和绘制到屏幕上的一切。

构造方法

```
new ig.Game()
```

默认的构造函数什么都不做。可以继承 ig.Game，并提供自己的代码。通常一个游戏的构造被称为通过 ig.main（ ）时，预载完成加载，或通过 ig.system.setGame（ ）当你想载入另一个游戏类。

目前正在运行的实例的一个游戏类提供 ig.game（小写），被引用实体等

属性

```
.autoSort
```

新的 1.19

是否要排序的实体。sortBy 发挥每一帧。默认是 false。

```
.backgroundMaps[]
```

一个实例的阵列 ig.BackgroundMap，控股的背景层。您可以使用。loadLevel（ ）方法加载这些保存与 Weltmeister 水平。

```
.backgroundAnims{}
```

声明：这是有点愚蠢可能改变未来版本:/

一个对象 ,指定特定的地形设置动画瓷砖。请注意 ,您必须设置装货前通过水平。loadLevel（ ） 。

```
//此设置任何背景图的两个砖（0 瓦和 5 瓦）
//媒体/ tiles.png 的地形设置是
var as = new ig.AnimationSheet( 'media/tiles.png', 16, 16 );
this.backgroundAnims = {
    'media/tiles.png': {
        0: new ig.Animation( as, 0.1, [0,1,2,3,4] ),
        5: new ig.Animation( as, 0.2, [5,6,7] )
    }
};
this.loadLevel( MyLevel );
```

```
.cellSize
```

检查和碰撞实体（见。checkEntities（ ） ）时的每一个细胞空间哈希的大小。如拇指规则，这应该是常见的实体大小的大约 4 倍。例如，如果您的实体大部分是 8x8 像素大小，设置。cellSize 32。不要担心太多，但。

默认是 64。

```
.clearColor
```

指定颜色的 CSS 串每帧绘图之前清零，如“＃00FF00”或“RGB（0,255,0）”。

1.17 补充：将此属性设置为空，如果你不想以清除之前绘制每一帧画面。

默认值是“＃000000”（黑色）。

```
.collisionMap
```

一个实例 ig.CollisionMap 或 ig.CollisionMap.staticNoCollision 如果你不希望针对 collisionMap 冲突。

默认是 ig.CollisionMap.staticNoCollision。

你可以使用游戏的.loadLevel（）方法来加载与 Weltmeister 保存水平。如果这个水平有一个名为地图“碰撞”会被自动设置为碰撞地图。

否则，使用构造函数创建一个碰撞地图：

```
// somewhere in your Game's init() method..
var data = [
    [1,2,6],
    [0,3,5],
    [2,8,1],
];
this.collisionMap = new ig.CollisionMap( 16, data );
```

```
.entities[]
```

数组在游戏世界中的所有实体。水平加载后（loadLevel（ ） ）。sortEntities（ ）被称为这个数组排序的实体。ZIndex 的。

你不应该需要直接操纵这个数组。使用游戏。spawnEntity（ ）和的。removeEntity（ ）方法来添加和删除实体。可用于检索由这个数组的实体。getEntityByName（ ）和。getEntitiesByType（ ）方法。

```
.gravity
```

重力（y 轴正加速度）适用于所有实体。注意：实体可以指定一个。gravityFactor 游戏的或多或少影响。重力。

默认为 0 -即没有重力。

```
.namedEntities{}
```

所有命名实体（与这些实体名称的属性集）的一个对象。游戏的。spawnEntity（ ）方法会自动将有一个名字，这个对象的实体。使用。getEntityByName（ ）来检索命名的实体。

```
.screen.x, .screen.y
```

像素在屏幕上的位置。把它看成是进入游戏世界的一个窗口的位置。

```
.sortBy
```

新的 1.19

指定使用排序功能。sortEntities（ ） 。可以是一个

- ig.Game.SORT.Z_INDEX
- ig.Game.SORT.POS_X
- ig.Game.SORT.POS_Y

默认是 Z_INDEX。排序 POS_Y 可以是有益的顶部下来 RPG 游戏，所以，在前面的字符总是重叠在后面。

方法

```
.checkEntities()
```

这种方法被称为每帧更新( )方法 ,并互相检查所有实体。它采用的单元尺寸 1 。cellSize 空间哈希这样做。

如果两个实体重叠，的静态 ig.Entity.checkPair（ ）函数被调用作为参数，这两个实体。这反过来又可以解决所有的实体与实体碰撞。

```
.draw()
```

这种方法被称为每一帧，并提请所有 BackgroundMaps 和实体。

```
.getEntityByName( name )
```

获得该人的姓名实体名称。实体阵列。返回不确定，如果无法找到实体。

```
.getEntitiesByType( type )
```

获取从给定类型的实体。实体数组的数组类型可以是一个字符串或实际的类对象。

例如：

```
var blobs = this.getEntitiesByType( 'EntityBlob' );
// or
var blobs = this.getEntitiesByType( EntityBlob );
```

请注意，子类 EntityBlob（例如 EntityBlobLarge）也将是类型 EntityBlob，因为 getEntitiesByType（ ）遍历整个类的原型链。

```
.loadLevel( levelObject )
```

加载给予 levelObject 从 Weltmeister 保存。

此方法将删除所有实体和背景和碰撞地图目前是否在游戏和重置屏幕坐标 0，0 之前建立新的实体和地图。

作为从 Weltmeister 保存 levelObject，结构如下：

```
{
    entities: [
        {type: "EntityClassName", x: 64, y: 32, settings: {}},
        {type: "EntityClassName", x: 16, y: 0, settings: {}},
        …
    ],

    layer: [
        {
            name: "background1",
            tilesetName: "media/tiles/biolab.png",
```

```
            repeat: false,
            distance: 1,
            tilesize: 8,
            foreground: false,
            data: [
                [1,2,6],
                [0,3,5],
                [2,8,1],
            ]
        },
        …
    ]
}
```

请注意，如果一层被命名为“碰撞”，它被认为是这个级别的 CollisionMap。

要小心，不要在一个中调用此方法更新（ ）周期（如通过触发） -切换所有的实体 ,而他们仍然被更新 ,并可能导致一些意想不到的行为。使用。loadLevelDeferred 代替。

```
.loadLevelDeferred( levelObject )
```

新的 1.16

一个呼叫。loadLevel（ ）推迟到一个更新周期结束。这可以确保实体“切换”，而他们仍然被更新。

```
.sortEntities()
```

重新排序。实体由他们的阵列。ZIndex 的正确设置的绘制顺序。

要小心，不要在一个中调用此方法更新（ ）周期（如通过触发） -重新排序，而

他们仍然被更新，并可能导致一些意想不到的行为实体。使用。sortEntitiesDeferred 代替。

新的 1.19

你可以指定排序功能，使用与.sortBy 财产。

```
.sortEntitiesDeferred()
```

新的 1.18

推迟 1 调用.sortEntities（ ）来更新周期结束。

```
.spawnEntity( type, x, y, settings )
```

创建一个指定类型(字符串或对象类)的新实体 ,并添加到游戏世界中的 X ，Y 。此方法调用实体类的构造，只是像往常一样。

实体。实体数组追加到结尾。你可以调用。sortEntities（ ）产卵后的实体，由他们诉诸的实体。ZIndex 的。

返回值是新创建的实体。

```
.removeEntity( entity )
```

删除从游戏世界的一个实体。通常情况下，这被称为从实体的。杀死（ ）方法。

```
.run()
```

这是主要的“运行循环”的游戏的方法。它被称为 FPS 次每秒，默认情况下，只要求游戏的更新（ ）。绘制（ ） 方法。

```
.update()
```

这种方法被称为为每个帧和更新所有实体，BackgroundMaps 和地形设置动画。

实体

在模块中的定义 impact.entity，从 ig.Class 继承

简介

```
// Create your own entity, subclassed from ig.Enitity
EntityPlayer = ig.Entity.extend({

    // Set some of the properties
    collides: ig.Entity.COLLIDES.ACTIVE,
    type: ig.Entity.TYPE.A,
    checkAgainst: ig.Entity.TYPE.B,

    size: {x: 16, y: 16},
    health: 50,

    // Load an animation sheet
    animSheet: new ig.AnimationSheet( 'media/player.png', 16, 16 ),

    init: function( x, y, settings ) {
        // Add animations for the animation sheet
        this.addAnim( 'idle', 0.1, [0,1,2] );
        this.addAnim( 'jump', 0.1, [3,4,5] );

        // Call the parent constructor
        this.parent( x, y, settings );
```

```
    }

    update: function() {
        // This method is called for every frame on each entity.
        // React to input, or compute the entity's AI here.

        if( ig.input.pressed('jump') ) {
            this.vel.y = -100;
            this.currentAnim = this.anims.jump.rewind();
        }

         // Call the parent update() method to move the entity
        // according to its physics
        this.parent();
    }
});
```

描述

在游戏世界的互动对象通常是从这个基地的实体类的子类。它提供动画，绘画和基本物理。子类从实体 ig.Entity 确保它可以被添加到游戏世界中，反应碰撞地图和其他实体，它可以用来在 Weltmeister。

这里列出的所有方法，可以在子类覆盖，并呼吁 this.parent（ ）如果需要的话。

构造方法

```
new ig.Entity( x, y, settings )
```

- X，Y 位置放置在游戏世界实体

- 设置一个 JavaScript 对象，其属性覆盖实体的默认属性

通常情况下，你应该通过创建实体 ig.Game 的 spawnEntity（ ）方法，该方法创建的实体，并把它添加到游戏世界。 设置对象覆盖一个特定的实体的属性。例如：

```
var settings = {health: 100, vel: {x: 200, y: 100}};
var myEnt = new EntityMyEntityClass( 0, 0, settings );
```

weltmeister 使设置对象的使用，存储额外的设置，为每个实体。

属性

```
.accel.x, .accel.y
```

当前加速被添加到实体的速度每秒。例如一个实体。vel.x 为 0 和。accel.x 10 后有一个。vel.x 的 100 十秒钟。

```
.animSheet
```

一个的实例 AnimationSheet，由实体的使用。addAnim（ ）方法。

```
.anims
```

控股实体的所有动画，通过创建。addAnim（ ）对象。例如：

```
this.addAnim( 'run', 0.1, [0,1,2] );
this.currentAnim = this.anims.run;
```

```
.bounciness
```

一个因素，说明实体与力量碰撞后会反弹。有了一个设置为 1 。反弹力，该实体将反弹回来击中了其他实体/碰撞地图以同样的速度。默认值为 0。

```
.checkAgainst
```

One of

- `ig.Entity.TYPE.NONE`
- `ig.Entity.TYPE.A`
- `ig.Entity.TYPE.B`
- `ig.Entity.TYPE.BOTH`

请参阅文件。类型的财产。

默认是 NONE。

```
.collides
```

One of

- `ig.Entity.COLLIDES.NEVER`
- `ig.Entity.COLLIDES.LITE`
- `ig.Entity.COLLIDES.PASSIVE`
- `ig.Entity.COLLIDES.ACTIVE`
- `ig.Entity.COLLIDES.FIXED`

这个属性决定与其他实体的实体碰撞。请注意，这是独立的碰撞对碰撞地图。

在主动与 Lite 或固定与任何冲突，只有疲软的实体移动，而另一个保持不变。在主动与积极，主动与被动的碰撞，两个实体所感动。Lite 或被动实体不与其他建兴或被动在所有实体碰撞。

固定与固定碰撞的行为是未定义的。

通常情况下，不重要的是游戏本身的实体，如颗粒，应碰撞 Lite 或从不。移动平台碰撞固定的，不受其他实体的运动。

在碰撞教程。

```
.currentAnim
```

一个的实例 ig.Animation，将由实体的绘制。绘制（ ）方法，或空，如果没有动画还没有被定义。

```
.friction.x, .friction.y
```

减速时要减去从实体每秒的速度。仅适用于.accel.* 0。

```
.gravityFactor
```

多少实体受重力在游戏类。一个为 0 。gravityFactor 将 使实体浮动，没有什么游戏的重力设置的问题。默认是 1。

```
.health
```

这个实体的健康。通过。receiveDamage（ ）方法通常是减少实体的健康。如果健康到达 0 时，该实体。杀（ ）方法被调用时，从游戏世界的实体。

```
.id
```

一个整数，代表了该实体的唯一的 ID。这个 ID 是由实体的设置。的 init( )方法。

请注意，该 ID 的实体。可能在每个级别的负载不同。

```
.last.x, .last.y
```

被称为实体的位置在最后一帧（更新（前））。这是用于实体与实体碰撞响应。

```
.maxVel.x, .maxVel.y
```

最大速度。实体的速度将这些值上限。

默认是 100。

```
.minBounceVelocity
```

如果低于这个阈值是实体的速度，也不会反弹。这是需要这样一个实体可以来一个完整的休息，而不是回一个微小的速度与弹跳。默认 40。

```
.offset.x, .offset.y
```

这个实体的动画绘制偏移。例如，如果你的实体的碰撞盒（，size.x ，。size.y）是 8 个像素宽，但您的动画帧的大小是 16 个像素宽，你指定一个。offset.x 4，转移碰撞盒 4 个像素右，中心动画帧。

```
.pos.x, .pos.
```

在游戏世界中的实体的立场。

```
.size.x, .size.y
```

实体的大小，以像素为单位。这是用于碰撞检测和响应。

默认的 x 和 y 的大小是 16。

```
.standing
```

布尔值，指出如果实体被搁在地面上（Y 轴）。

```
.slopeStanding
```

新的 1.19

对象界定在坡角。地位属性仍设置。默认是

```
slopeStanding: {min: (44).toRad(), max: (136).toRad() }
```

```
.type
```

One of

- ig.Entity.TYPE.NONE
- ig.Entity.TYPE.A
- ig.Entity.TYPE.B

通过。类型属性，实体可以组织成两组（或无）之一。这是有用的。checkAgainst 财产与合作。

当两个实体重叠。checkAgainst 一个实体的属性相匹配的其他类型，前者实体的检查（ ）方法与后者的实体作为参数调用。

例如 ,你可以给所有友好实体的 A 型 ,并设立的一切敌对实体。checkAgainst 的。在检查（ ）方法的敌对实体，然后你可以给友好的实体损坏。

默认是 NONE。

```
.vel.x, vel.y
```

流速每秒的像素。

```
.zIndex
```

绘制顺序。具有较高的实体。的 ZIndex 将获得最后绘制。请注意，实体，默认情况下，只有一次后级负载（排序 ig.Game 。loadLevel（ ） ）。产卵后的实体，如果他们将在比赛结束时追加实体阵列，从而得出最后。打电话：ig.Game 的 。sortEntitiesDeferred（ ）重新对它们进行排序。例如：

```
// Spawn 50 "particle" entities. Set a negative zIndex, so they will
// be drawn first and occluded by all other entities
for( var i = 0; i < 50; i++ ) {
    var e = ig.game.spawnEntity( EntityMyParticle, x, y );
    e.zIndex = -10;
}
// Re-sort Entities
ig.game.sortEntitiesDeferred();
```

```
._wmScalable
```

无论是实体的大小是可以改变的在 Weltmeister。默认是 false。

```
._wmDrawBox
```

是否 Weltmeister 应制订一个盒子，这个实体。在游戏中无形的实体，这是有用

的。默认是 false。

```
._wmBoxColor
```

当_wmDrawBox 是 true，此属性指定框的颜色，将在 Weltmeister 的绘制。使用 CSS 的字符串，如'＃FF00FF'或'RGBA（255,0,255,0.5）“ 。

方法

```
.addAnim( name, frameTime, sequence, [stop] )
```

指定一个实体的动画。animSheet。这种方法本质上是一个速记动画的构造。

- name 命名实体的新动画的名称。anims 对象。
- frameTime 将显示在几秒钟的时间动画的每一帧
- sequence 一个数组序列动画的帧号。
- stop 布尔。确定是否应停止在最后一帧动画，或无限期地重复。为默认假。

如果实体。currentAnim 是空的，新创建的动画设置作为.currentAnim。

你可以访问通过此方法创建一个动画。anims 财产。例如：

```
this.addAnim( 'run', 0.1, [0,1,2,3,4] );
this.anims.run.flip.x = true;
```

```
.angleTo( other )
```

返回的角度，从这个实体的中心，其他实体的中心。

```
.check( other )
```

这种方法被称为如果其他实体和其他实体的类型与实体重叠匹配实体。

checkAgainst 财产。

```
.collideWith( other, axis )
```

这种方法被称为。轴与另一实体实体碰撞时，碰撞轴：要么'X'或'Y' 。

```
.draw()
```

自动调用每个帧的游戏。绘制当前动画。

```
.distanceTo( other )
```

返回从这个实体的中心，以像素为单位的绝对距离其他实体的中心。

```
.handleMovementTrace( res )
```

默认的 update（ ）方法的实体不反对运动轨迹 CollisionMap 的。。

handleMovementTrace（ ）方法接收此跟踪的结果，并采取相应的行动。

你可以覆盖此方法退出碰撞或创造一些额外的行为。

举例来说，如果你想播放声音时，您的播放器实体击中地板具有一定的速度，你可以覆盖。handleMovementTrace（ ）和检查碰撞和速度。

```
EntityPlayer = ig.Entity.extend({
```

```
…

handleMovementTrace: function( res ) {
    if( res.collision.y && this.vel.y > 32 ) {
        this.soundBounce.play();
    }

    // Continue resolving the collision as normal
    this.parent(res);
},

…

});
```

```
.kill()
```

删除从游戏世界的实体。这种方法被称为。 receiveDamage（ ）实体的健康，如果是 0 或更少。

```
.ready()
```

新的 1.19

这个函数被调用一次后水平完全加载，所有的实体存在。用它来初始化等实体之间的连接。这个函数的默认实现不做任何事情。

```
.receiveDamage( amount, from )
```

substracts 金额从实体。健康和调用。杀（ ）实体的新的健康，如果低于或等于

0。

这个函数应该被称为损害给予的实体 ,从参数。然而 ,这个参数 ,目前没有使用。

```
.touches( other )
```

返回 true，如果这个实体接触的其他实体，否则为 false。

```
.update()
```

这种方法被称为每一帧游戏绘图之前。根据它的速度，摩擦，反弹力等，并更新当前动画基地实体。更新( )方法将实体。这是该实体的地方添加你自己的逻辑。如果覆盖此方法，你仍然可以调用基实体的更新（ ）与方法 this.parent（ ） 。例如，看到的梗概。

输入

在模块中定义 impact.input，从 ig.Class 继承

简介

```
// On game start
ig.input.bind( ig.KEY.UP_ARROW, 'jump' );


// In your game's or entity's update() method
if( ig.input.pressed('jump') ) {
    this.vel.y = -100;
}
```

描述

ig.Input 处理所有的键盘和鼠标输入。

可以绑定按键的具体行动，然后问，如果这些行动之一，目前持有（状态（ ）），或者只是按下后的最后一帧（按下（ ））。

请注意，将被自动一个实例 ig.Input 的创建 ig.input（小写）由 ig.main（ ）函数。

属性

```
.accel.x, .accel.y, .accel.z
```

新的 1.18

当前加速度米/秒²设备。这将 0 开始的设备，没有一个加速度。

调用.initAccelerometer（ ）开始捕捉加速度计输入。

```
.mouse.x, .mouse.y
```

像素您在游戏屏幕上的鼠标光标的位置。请注意，这个位置至少一个鼠标按钮绑定到一个动作时，只更新。

如果你还没有绑定任何鼠标按钮，你可以调用自己 initMouse（ ）一次，开始捕获鼠标输入。

方法

```
.bind( key, action )
```

绑定一个键盘或鼠标按钮的行动。行动是一个字符串或整数，确定在游戏中的行动。调用此为每个按钮的功能和动作对你的游戏开始。

键可以是任何的 ig.KEY。*。

几个按钮，可以绑定到相同的动作，但一个按钮不能被绑定到几个动作。

```
// Binds both, UP_ARROW and SPACE, to the same action
ig.input.bind( ig.KEY.UP_ARROW, 'jump' );
ig.input.bind( ig.KEY.SPACE, 'jump' );
```

```
.bindTouch( selector, action )
```

绑定的 HTML 元素发出指定的动作，当元素被感动。这将仅适用于设备实现的 touchstart 和 touchend 事件-目前 iOS 和 Android 设备。

所指定的元素选择器必须已经存在。它不会被创建的影响。选择只能是一个 id 选择，例如“＃键”。

```
//绑定的<div id="button1"> </ DIV>元素的“跳”动作
ig.input.bindTouch( '#button1', 'jump' );
```

```
.initAccelerometer()
```

新的 1.18

开始捕捉加速度计输入。将可在加速。accel.x。accel.y。accel.z。这将被忽略，

没有一个加速度的设备。

```
.initMouse()
```

开始捕获鼠标输入。将可在当前的鼠标位置。mouse.x。mouse.y。

这种方法被自动调用尽快作为一个鼠标按钮已经绑定。

```
.pressed( action )
```

返回 true，如果绑定到指定的按钮，一个动作，只是压下来，否则 false。

相反.state() ，此方法只返回真正的一次每次按下按钮。你可以使用这个例子，

如果你希望用户反复按下一个按钮来拍摄，而不是“连射”。

```
.released( action )
```

新的 1.19

返回 true，如果绑定到指定的按钮，一个动作刚刚发布，否则 false。

```
.state( action )
```

返回 true，如果一个动作绑定到指定的按钮，目前压下去，否则假。

```
.unbind( key )
```

解除其目前的行动从键盘或鼠标按钮。

键可以是任何的 ig.KEY。*。看到下面的列表。

```
.unbindAll()
```

解除所有键盘，鼠标和触摸按钮。

鼠标和键盘的按钮名称

```
ig.KEY.MOUSE1

ig.KEY.MOUSE2

ig.KEY.MWHEEL_UP

ig.KEY.MWHEEL_DOWN

ig.KEY.BACKSPACE

ig.KEY.TAB

ig.KEY.ENTER

ig.KEY.PAUSE

ig.KEY.CAPS

ig.KEY.ESC

ig.KEY.SPACE

ig.KEY.PAGE_UP

ig.KEY.PAGE_DOWN

ig.KEY.END

ig.KEY.HOME

ig.KEY.LEFT_ARROW

ig.KEY.UP_ARROW

ig.KEY.RIGHT_ARROW

ig.KEY.DOWN_ARROW

ig.KEY.INSERT

ig.KEY.DELETE

ig.KEY._0

ig.KEY._1

ig.KEY._2

ig.KEY._3

ig.KEY._4
```

```
ig.KEY._5
ig.KEY._6
ig.KEY._7
ig.KEY._8
ig.KEY._9
ig.KEY.A
ig.KEY.B
ig.KEY.C
ig.KEY.D
ig.KEY.E
ig.KEY.F
ig.KEY.G
ig.KEY.H
ig.KEY.I
ig.KEY.J
ig.KEY.K
ig.KEY.L
ig.KEY.M
ig.KEY.N
ig.KEY.O
ig.KEY.P
ig.KEY.Q
ig.KEY.R
ig.KEY.S
ig.KEY.T
ig.KEY.U
ig.KEY.V
ig.KEY.W
ig.KEY.X
ig.KEY.Y
ig.KEY.Z
```

```
ig.KEY.NUMPAD_0

ig.KEY.NUMPAD_1

ig.KEY.NUMPAD_2

ig.KEY.NUMPAD_3

ig.KEY.NUMPAD_4

ig.KEY.NUMPAD_5

ig.KEY.NUMPAD_6

ig.KEY.NUMPAD_7

ig.KEY.NUMPAD_8

ig.KEY.NUMPAD_9

ig.KEY.MULTIPLY

ig.KEY.ADD

ig.KEY.SUBSTRACT

ig.KEY.DECIMAL

ig.KEY.DIVIDE

ig.KEY.F1

ig.KEY.F2

ig.KEY.F3

ig.KEY.F4

ig.KEY.F5

ig.KEY.F6

ig.KEY.F7

ig.KEY.F8

ig.KEY.F9

ig.KEY.F10

ig.KEY.F11

ig.KEY.F12

ig.KEY.SHIFT

ig.KEY.CTRL

ig.KEY.ALT

ig.KEY.PLUS
```

```
ig.KEY.COMMA

ig.KEY.MINUS

ig.KEY.PERIOD
```

定时器

在模块中定义 impact.timer，从 ig.Class 继承

简介

```
var timer = new ig.Timer();


// Difference between current time and the time .set()

timer.delta(); // => 0


// Set the timer to 5 seconds in the future

timer.set( 5 );

timer.delta(); // => -5


// ... one second later

timer.delta(); // => -4


// ... 5 more seconds later

timer.delta(); // => 1
```

描述

该 ig.Timer 具有两个不同的操作模式。你可以得到当前的时间和定时器（三角洲（）之间的差异）的目标时间（由设置构造或设置（）。 ）或刚刚获得当前 滴答 -以来的最后一次通话时间打勾（ ）。

所有的定时器都只有每帧更新一次。所以调用。三角洲（ ）一个定时器多次在一帧，将导致在相同的返回值。

ig.Timer 不测量实时，但比赛时间。游戏时间可以比实时进步慢，当浏览器无法执行游戏循环几次每秒（默认 20 ）。看到 ig.Timer.maxStep。

如何快速比赛时间的推移也取决于对全球 ig.Timer.timeScale 财产。

要知道，所有的时间间隔是在几秒钟内，反对的 JavaScripts 毫秒。

构造方法

```
new ig.Timer( [seconds] )
```

相对秒的目标时间。默认值为 0。

方法

```
.delta()
```

返回在几秒钟的时间，相对的定时器目标时间。见。设置（ ） 。

如果在未来的目标时，这个函数的返回值将是负数（例如-5 5 秒，在未来）。如果目标时间是在过去，返回值将是积极的。它本质上返回的时间 “，因为” 目标的时间。

例如，如果要测量多久了球员来完成你的游戏水平，只是创建一个新的定时器启动时的水平，并调用。三角洲（ ）时的水平结束。

```
// at level start
var playTimer = new ig.Timer(); // no "target" time given = NOW.


// ... at level end
var timeNeeded = playTimer.delta();


// reset the timer at the start of the next level
playTimer.reset();
```

```
.reset()
```

重置这个计时器的开始时间。这并不重置目标的时间。

```
var timer = new ig.Timer( 5 );


// ... one second later
timer.delta(); // => -4


timer.reset();
timer.delta(); // => -5
```

```
.set( [seconds] )
```

重置这个计时器的开始时间，并设置相对目标的时间。默认值为 0。

```
var timer = new ig.Timer( 5 );


// ... one second later
timer.delta(); // => -4


timer.set( 3 );
timer.delta(); // => -3
```

```
.tick()
```

获取以来的最后一次通话时间。打勾（此计时器）。请注意，因为 ig.Timer 代表游戏时间，它并不在一个单一的框架更新。例如：

```
var timer = new ig.Timer();


timer.tick(); // => 0
doSomethingVeryComplicatedThatTakesAFewSeconds( andBlocksEverything );
timer.tick(); // => 0
```

一个全球性的刻度，每帧更新，还提供在 ig.system.tick。

全局属性

```
ig.Timer.maxStep
```

最大的时间步长。游戏的时间绝不会比这每帧前进。默认值是 0.05，这等于最低 20 帧的帧速率。如果浏览器是无法运行在至少 20 帧，游戏基本上放缓。

```
ig.Timer.timeScale
```

比赛时间的因素。有了一个价值 0.5 游戏将在慢动作运行，2，它将运行在两倍的速度。默认是 1。

地图

模块中的定义 impact.map，从 ig.Class 继承

简介

```
// Create a map
var data = [
    [1,2,6],
    [0,3,5],
    [2,8,1],
];
var map = new ig.Map( 16, data );

// 40, 20 are the pixel coordinates for the tile to set
// So this essentially does data[2][1] = 8;
map.setTile( 40, 20, 8 );

// Gets the same tile we just set, even though the
// pixel coordinates are not the exact same
var tile = map.getTile( 46, 18 );
tile; // => 8
```

描述

ig.Map 是基类 ig.BackgroundMap 和 ig.CollisionMap。它仅提供了基本的访问在地图数据的瓷砖。

你通常不会需要创建一个实例 ig.Map。

构造方法

```
new ig.Map( tilesize, data )
```

- `tilesize`一个单一的瓷砖的宽度和高度，以像素

- `data`数据一个二维数组（数组的数组）作为地图数据

属性

```
.data
```

这张地图上，通过构造二维数据阵列。

```
.tilesize
```

以像素为单位的单一瓷砖的宽度和高度，通过构造。

```
.width, .height
```

瓷砖地图的宽度和高度。

方法

```
.getTile( x, y )
```

获取的像素坐标 X ， Y 的瓷砖。如果躺在这张地图之外的坐标，则返回 0。

```
.setTile( x, y, tile)
```

设置在像素坐标 X ， Y 的 瓷砖给定的瓷砖。什么也不做，如果躺在这张地图之外的坐标。

CollisionMap

在模块中定义地图 impact.collision，从 ig.Map 继承

简介

```
//创建 CollisionMap
var data = [
    [1,2,6],
    [0,3,5],
    [2,8,1],
];
var collision = new ig.CollisionMap( 16, data );


//做一个跟踪
var res = collision.trace( x, y, vx, vy, objectWidth, objectHeight );
if( res.collision.x || res.collision.y ) {
    //跟踪在地图相撞(res.pos.x, res.pos.y)
}
```

描述

一个 ig.Collision 需要一个 2D tilemap，并允许跟踪碰撞反对。

类的 ig.Game “ 。loadLevel（ ）方法需要，从 Weltmeister 保存数据，并创建一个 CollisonMap 如果存在。

默认情况下，所有的实体跟踪对 ig.game.collisionMap 作为其更新周期的一部分。

构造方法

```
new ig.CollisionMap( tilesize, data, [tiledef] )
```

- `tilesize`一个单一的瓷砖的宽度和高度，以像素
- `data`表明在哪里画一个二维数组（数组的数组）从地形设置的瓷砖。

新的 1.19

可选 tiledef 参数指定瓷砖的定义，使用。这定义这砖是斜坡的等默认 ig.CollisionMap.defaultTileDef 的。传递一个空对象（{} ）只有固体/固体瓷砖。

属性

```
.firstSolidTile
```

在 1.19 中删除

第一瓦数被认为是坚实的。缺省值 1。

```
.lastSolidTile
```

在 1.19 中删除

最后的瓦数被认为是坚实的。默认 255。

方法

```
.trace( x, y, sx, sy, objectWidth, objectHeight )
```

是否从跟踪(X ,Y)(X + Y + SY SX)为具有指定的对象 objectWidth 和 objectHeight

的。此方法返回在下列表格中的“跟踪结果”：

```
{
    collision: { x: false, y: false, slope: false },
    pos: { x: 0, y: 0 }, // Resulting object position
    tile: { x: 0, y: 0 } // Tile for the collision
}
```

新的 1.19

要么是假的 collision.slope 属性（如果有没有斜坡碰撞）或对象的一个属性 X，Y -坡方向- NX，NY -斜坡正常。

图片

在模块中定义 impact.image，从 ig.Class 继承

简介

```
// Load an image
var img = new ig.Image( 'player.png' );
// Draw the whole image
img.draw( x, y );
// Draw one 'tile' of the image
// (tile 3, with a tilesize of 16x16 pixels)
img.drawTile( x, y, 3, 16 );
```

描述

ig.Image 是围绕图像资源（PNG，GIF 或 JPEG）的包装。它需要加载和缩放源图像。你可以借鉴的整体形象（。平局（ ） ）或只是它的一个瓦（drawTile（ ） ）。

大部分时间，你应该不需要直接绘制 ig.Image，而是使用 ig.Animation 或 ig.BackgroundMap。

构造方法

```
new ig.Image( filename )
```

文件名 的图像文件的路径和名称被加载。

请注意，图像缓存。这意味着，如果您创建两个图像具有相同的源文件，构造函数返回相同的实例。例如：

```
var img1 = new ig.Image( 'player.png' );
var img2 = new ig.Image( 'player.png' );


img1 == img2; // => true
```

加载图像后，它会自动缩放（近邻）根据 ig.system.scale 如有必要。如果发动机没有完全加载所有所需的模块（记为 ig.ready），图像被载入 preloader 的资源。

属性

```
.data
```

加载图像时的实际图像资源。这是一个 <img> 或（ ，如果 ig.system.scale 是不是 1）<canvas>元素。

```
.failed
```

true 时无法加载图像资源（如 404 或 403 错误）

```
.path
```

true 已加载的图像资源，否则假

```
.path
```

图像文件的路径和名称传递给构造。

```
.width, .height
```

（未缩放）像素图像的大小。

方法

```
.draw( targetX, targetY, [sourceX], [sourceY], [width], [height]
)
```

绘制在整个的形象 targetX，targetY，或当 sourceX，sourceY 和宽度和高度给出了它的一部分。

```
.drawTile( targetX, targetY, tile, tileWidth, [tileHeight], [flip
X],[flipY] )
```

绘制指数与一个单一的瓷砖瓷砖在 targetX，targetY。

计算与 tileWidth 和 tileHeight 指定瓷砖的位置。如果 tileHeight 被省略，它被假定是一样 tileWidth 的。

flipX 和 flipY 是布尔值，表示如果瓷砖应制定翻转（“镜像”）在给定的轴。

动画

在模块中定义 impact.animation，从 ig.Class 继承

简介

```
// Create animation
var animSheet = new ig.AnimationSheet( 'sheet.png', 16, 16 );
var anim = new ig.Animation( animSheet, 0.1, [0,1,2,3,2,1] );


// Update animation to current frame
anim.update();


// Draw current frame
anim.draw( x, y );
```

描述

一个 ig.Animation 对象的动画实体或背景图瓷砖。由动画指 定 frameTime 和序列帧动画表-所有动画帧的图像-绘制。

在大多数情况下，你不应该需要创建自己的动画对象，而是使用了 ig.Entity 的 。addAnim（ ）方法。

阅读更多的动画教程。

构造方法

```
new ig.Animation( sheet, frameTime, sequence, [stop] )
```

- `sheet` ig.AnimationSheet 对象，指定要使用的图像和每帧的宽度和高度

每帧的显示时间，以秒

- `frameTime` 序列整数，一个数组指定实际的动画帧
- `sequence` 一个整数数组，指定实际的动画帧
- `stop` 一个布尔值 ,指示是否停止动画时 ,它已经达成的最后一帧。默认为 false。

属性

```
.alpha
```

阿尔法透明度的动画从 0 到 1。默认为 1（完全不透明）。

```
.angle
```

新的 1.15

旋转弧度动画。指定旋转中心。支点。

```
.flip.x, .flip.y
```

布尔值，指示是否应绘制动画在 x 和/或 Y 轴翻转

```
.frame
```

当前帧的数量，由设置更新（ ）方法

```
.loopCount
```

一个整数，指定多久动画已发挥自上次调用通过倒带（ ） 。

请注意，即使已停止在最后一帧的动画，动画，仿佛仍在运行更新。loopCount。

```
.pivot.x, .pivot.y
```

新的 1.15

中心时使用。角度的旋转。默认是一半的宽度和高度的 AnimationSheet。，即动画帧的中心。

```
.tile
```

目前瓷砖，设置动画表`.update()`方法

方法

```
.draw( x, y)
```

绘制当前动画帧在给定的屏幕坐标

```
.gotoFrame( f )
```

跳跃帧号 f

```
.gotoRandomFrame()
```

跳转到一个随机帧

```
.rewind()
```

倒带动画的第一帧和复位。loopCount

此方法返回动画本身。动画之间切换时，这是非常有用的。例如：

```
// In an entities .update() method
this.currentAnim = this.anims.jump.rewind();
```

```
.update()
```

更新的动画当前帧，根据其 frameTime

AnimationSheet

在模块中定义 impact.animation，从 ig.Class 继承

简介

```
// Create animation
var animSheet = new ig.AnimationSheet( 'sheet.png', 16, 16 );
var anim = new ig.Animation( animSheet, 0.1, [0,1,2,3,2,1] );

// Update animation to current frame
anim.update();

// Draw current frame
anim.draw( x, y );
```

描述

一个 ig.Animation 对象的动画实体或背景图瓷砖。由动画指 定 frameTime 和序列帧动画表-所有动画帧的图像-绘制。

在大多数情况下 ,你不应该需要创建自己的动画对象 ,而是使用了 ig.Entity 的 。addAnim（ ）方法。

阅读更多的动画教程。

构造方法

```
new ig.Animation( sheet, frameTime, sequence, [stop] )
```

- `sheet` ig.AnimationSheet 对象，指定要使用的图像和每帧的宽度和高度

每帧的显示时间，以秒

- `frameTime` 序列整数，一个数组指定实际的动画帧
- `sequence` 一个整数数组，指定实际的动画帧
- `stop` 一个布尔值 ,指示是否停止动画时 ,它已经达成的最后一帧。默认为 false。

属性

```
.alpha
```

阿尔法透明度的动画从 0 到 1。默认为 1（完全不透明）。

```
.angle
```

新的 1.15

旋转弧度动画。指定旋转中心。支点。

```
.flip.x, .flip.y
```

布尔值，指示是否应绘制动画在 x 和/或 Y 轴翻转

```
.frame
```

当前帧的数量，由设置更新（ ）方法

```
.loopCount
```

一个整数，指定多久动画已发挥自上次调用通过倒带（ ） 。

请注意，即使已停止在最后一帧的动画，动画，仿佛仍在运行更新。loopCount。

```
.pivot.x, .pivot.y
```

新的 1.15

中心时使用。角度的旋转。默认是一半的宽度和高度的 AnimationSheet。，即动画帧的中心。

```
.tile
```

目前瓷砖，设置动画表 `.update()` 方法

方法

```
.draw( x, y)
```

绘制当前动画帧在给定的屏幕坐标

```
.gotoFrame( f )
```

跳跃帧号 f

```
.gotoRandomFrame()
```

跳转到一个随机帧

```
.rewind()
```

倒带动画的第一帧和复位。loopCount

此方法返回动画本身。动画之间切换时，这是非常有用的。例如：

```
// In an entities .update() method
this.currentAnim = this.anims.jump.rewind();
```

```
.update()
```

更新的动画当前帧，根据其 frameTime

字体

模块中的定义 impact.font，从 ig.Image 继承

简介

```
var font = new ig.Font( 'font.png' );

font.draw( 'Some text', x, y, ig.Font.ALIGN.RIGHT );
```

描述

ig.Font 对象加载一个特殊格式的字体图像，让你用它绘制的文本。

ig.Font 使用位图字体-它从一个形象，而不是使用了 Canvas API，字形 fillText（）方法。

字体图像中的像素的最底部行指定为每个单个字符的字符宽度。运行的非透明像素代表一个字符，其宽度。因此，字体图像中的所有字符必须在同一行。

您可以使用字体工具生成一个从任何已安装的系统字体的字体图像。

构造方法

```
new ig.Font( filename )
```

`filename` 要加载的字体图像的路径和名称。

请注意的是，类似 ig.Image，字体缓存。这意味着，如果你创建了两种字体相同的字体形象，构造函数返回相同的实例。例如：

```
var font1 = new ig.Font( 'font.png' );
var font2 = new ig.Font( 'font.png' );


font1 == font2; // => true
```

属性

```
.firstChar
```

第一个 ASCII 字符，这是目前在字体图像。默认值是 32（空格）。

```
.height
```

新的 1.16

以像素为单位的字体高度。这是源图像的高度等于占的像素的最后一行。

方法

```
.draw( text, x, y, [align] )
```

此方法绘制字符串文本在 X ， Y 。

可选的对齐参数可以是一个：

- `ig.Font.ALIGN.LEFT`
- `ig.Font.ALIGN.RIGHT`
- `ig.Font.ALIGN.CENTER`

如果省略，左边的是假设。

新的 1.19

换行符（'\ n' ）现在正确的荣幸和绘图继续高度像素进一步下降

```
.widthForString( text )
```

新的 1.16

返回给定的文本字符串，这个字体的像素宽度。

BackgroundMap

在模块中定义地图 impact.background，从 ig.Map 继承

简介

```
// Create BackgroundMap
var data = [
    [1,2,6],
    [0,3,5],
    [2,8,1],
];
var bg = new ig.BackgroundMap( 16, data, 'media/tileset.png' );

// Move
bg.setScreenPos( 25, 10 );

// Draw
bg.draw();
```

描述

提请 ig.BackgroundMap 地形设置，其二维数据阵列表示的瓷砖。

地形设置图像的大小必须在 tilesize 的倍数，否则影响会感到困惑与编号瓷砖。

32 地形设置图像的宽度和高度的 tilesize 如：32，64，96，128 应该是...

你可以手工创建一个 BackgroundMap，或使用的 ig.Game 类“ 。loadLevel ( ) 方法，这需要从 Weltmeister 保存数据，并根据它创建 BackgroundMaps。

的 BackgroundMaps 可以预先渲染成“块”，而不是单独绘制每瓦。这加快了拉丝工艺的巨大，特别是在移动设备上。然而，预渲染的模式，需要更多的内存和不支持地形设置动画。

构造方法

```
new ig.BackgroundMap( tilesize, data, tileset )
```

- `tilesize` 一个单一的瓷砖的宽度和高度，以像素
- `data` 表明在哪里画一个二维数组(数组的数组)从地形设置的瓷砖。请注意，瓷砖的编号是由一 ,即 0 瓷砖画什么和瓦 1 绘制的地形设置图像的零级(一) 瓷砖。
- `tileset` 地形设置地点的地形设置使用的图像文件或实例 ig.Image

属性

```
.anims{}
```

对象指定某些砖的动画。动画不由 BackgroundMap 更新，但外部必须更新。一般 ig.Game，照顾-看到 ig.Game 的 backgroundAnims 财产。

默认情况下，这是一个空对象（{} ）表示没有动画。

```
var data = [
    [1,2,6],
    [0,3,5],
    [2,8,1],
];
var bg = new ig.BackgroundMap( 16, data, 'media/tileset.png' );
```

```
// sets tile number 5 of the BackgroundMap to be animated
var as = new ig.AnimationSheet( 'media/tiles.png', 16, 16 );
bg.anims[5] = new ig.Animation( as, 0.1, [0,1,2,3,4] );
```

```
.chunkSize
```

块的宽度和高度预渲染 BackgroundMaps 的像素。默认是 512。见。的 PreRender。iOS 设备似乎支持的块大小为 4096，但测试表明没有进一步的性能增益比 512 更大的块大小。

```
.debugChunks
```

.prerender 和 debugChunks 如果是真，绘制（ ）方法将吸引周围每块粉红色的轮廓。

```
.distance
```

距离因素的 BackgroundMap，说明如何快速的地图卷轴。setScreenPos（ ）被调用时。随着距离 1，地图滚动到屏幕上同步。与的距离为 2 的地图卷轴速度的一半，使得它看上去更远的背景。默认是 1

```
.foreground
```

新的 1.17

这层是否会被绘制在前面的所有实体。这是由 Weltmeister 自动设置。默认是假

的。

```
.preRender
```

如果设置为 true ,将在下次调用绘制( )渲染成块的整个背景图。块大小的像素，然后使用这些块，绘制地图。

预渲染的地图不能是动画，但通常是绘制速度远远超过瓷砖的地图。

默认是 false。

```
.repeat
```

一个布尔值，指示是否绘制的地图包的边缘。这是不断重复的二次 BackgroundMaps 有用。

```
.scroll.x, .scroll.y
```

滚动在屏幕 BackgroundMap 的位置坐标与地图的距离。 alreday 因素英寸只应予以使用。setScreenPos（ ）设置滚动位置的地图。

```
.tiles
```

一个实例代表的地形设置，使用绘图，构造或者 ig.Image 。setTileset（ ）

```
.tilesetName
```

加载的地形设置图像的位置（例如`'media/tileset.png'`）

方法

```
.draw()
```

绘制在当前的 BackgroundMap 滚动位置。如果是真实的。的 PreRender，绘制地图的预渲染块，否则瓷砖瓷砖的地图绘制。

```
.setScreenPos( x, y )
```

设置的 x 和 y 的屏幕上的位置。将设置 acording 其距离的 BackgroundMap 的滚动条的位置。

```
.setTileset( tileset )
```

设置一个新的地形设置。地形设置的图像文件或实例的位置 ig.Image 地形设置。

声音

模块中的定义 impact.sound，从 ig.Class 继承

简介

```
var sound = new ig.Sound( 'shoot.ogg' );
sound.play();
```

描述

一个实例 ig.Sound 代表一个声音文件，用来作为背景音乐或游戏音效。

构造方法

```
new ig.Sound( filename, [multiChannel] )
```

- `filename` 要加载的声音文件的路径和名称。
- `multiChannel` 是否应建立多个“通道”，同时发挥。通过虚假的音乐或声音文件只播放一次。默认值是 true。

你所有的声音文件必须在目前的 Ogg Vorbis 和 MP3 格式。它们的文件名 是相同的文件扩展名除外。将加载真正的文件名 可以从给定的文件名 不同，，作为 ig.SoundManager 文件扩展名决定。也看到 ig.Sound.use。

例如，“跳”的声音，你有下列文件：

```
media/sounds/jump.ogg
media/sounds/jump.mp3
```

然后可以用下面的语句之一载入

```
//这 3 条线都做同样的。文件扩展名（。OGG，MP3）
//的 SoundManager 基于浏览器的功能，决定
//使用任何语法你最喜欢的
var sound = new ig.Sound( 'media/sounds/jump.ogg' );
var sound = new ig.Sound( 'media/sounds/jump.mp3' );
var sound = new ig.Sound( 'media/sounds/jump.*' );
```

属性

```
.path
```

图像文件的路径和名称传递给构造。

```
.volume
```

这声音文件的体积 0.0 到 1.0。这将成倍的 ig.soundManager.volume 前播放声音。默认是 1。

方法

```
.play()
```

播放声音文件。如果不加载的声音文件作为多声道(见构造函数),它已在播放，声音从一开始就重新启动，否则播放的声音同时在另一个通道。

```
.stop()
```

停止播放声音

全局属性

```
ig.Sound.enabled
```

您可以禁用一切听起来音乐将此属性设置为假前调用 ig.main ( ) 。默认的是真实的。

当 ig.Sound.enabled 是假的，没有健全的文件将被载入或播放。这是用于移动设备，目前仍然有广阔的问题，在浏览器中的声音播放。

```
//如果禁用移动设备的所有声音
if( ig.ua.mobile ) {
    ig.Sound.enabled = false;
}
```

```
// Start the game

ig.main(…)
```

```
ig.Sound.channels
```

新的 1.17

设置每个声音文件创建副本的数量。有更多的拷贝，意味着可以播放的声音同时多次。这必须设置在打电话之前 ig.main（ ）

默认值为 4。

```
ig.Sound.FORMAT{}
```

新的 1.17

指定一个对象的各种声音格式的文件扩展名和 MIME 类型。这是用来与配合 ig.Sound.use。目前实施的是：

- `ig.Sound.FORMAT.OGG`
- `ig.Sound.FORMAT.MP3`
- `ig.Sound.FORMAT.M4A`
- `ig.Sound.FORMAT.WEBM`
- `ig.Sound.FORMAT.CAF`

```
ig.Sound.use[]
```

新的 1.17

一个阵列 ig.Sound.FORMAT 属性来指定测试支持的声音格式。第一个支持的格式，将加载的所有声音文件的文件扩展名决定。

默认 true

```
[ig.Sound.FORMAT.OGG, ig.Sound.FORMAT.MP3]
```

也就是说 如果浏览器支持 OGG / Vorbis 格式 将影响尝试加载 filename.ogg 的。

如果不支持 OGG / Vorbis 是，支持 MP3 测试和 filename.mp3 将被载入。如果没有格式的支持，是完全禁止的声音。

目前，支持 OGG / Vorbis 是由 Firefox，Opera 和 Chrome。支持 Safari，Chrome 和 IE9 的 MP3 和 M4A。

音乐

模块中的定义 impact.sound，从 ig.Class 继承

简介

```
ig.music.add( 'music/track1.ogg' );
ig.music.add( 'music/track2.ogg' );
ig.music.add( 'music/track3.ogg' );

ig.music.volume = 0.5;
ig.music.play();
```

描述

ig.Music 提供能力发挥在有序或随机的时尚背景音乐列表。

会自动创建一个实例 ig.Music 由 ig.main（ ）函数在 ig.music（小写 ）。

属性

```
.currentIndex
```

该指数在目前的轨道。轨道阵列。

```
.currentTrack
```

的实例 ig.Sound，当前正在播放。空，如果是空的。轨道。

```
.loop
```

Wheter 当前曲目应循环，而不是跳转到下一首曲目。如果是只有一个轨道。轨道阵列，此属性将被忽略。默认是假的。

```
.random
```

如果真正的，以随机顺序播放曲目。默认是假的。

```
.tracks[]
```

要播放一个实例数组 ig.Sound。

```
.volume
```

所有音乐的音量，跟踪从 0.0 到 1.0。默认是 1。

方法

```
.add( track, [name] )
```

增加了给定的轨道.`track`阵列和集。currentTrack 这一个，如果是无以复加的第一首曲目。

`track`可以是一个实例的 ig.Sound 或一个声音文件的文件名 和路径。你应该载入所有的声音文件作为类的属性，以确保它们是由 Preloader 的加载。看到与资产的工作。

新的 1.19

使用可选的 name 参数指定曲目名称。可以播放.play（ ）方法命名为轨道。

```
.fadeOut( time )
```

线性淡出超过当前曲目时间秒。

淡出是完整的，当.stop()被称为这条赛道的体积是它的原始音量复位。

```
.next()
```

停止当前的轨道，并跳转到在未来的轨道。轨道阵列。如果是真实的。随机，随机选择下一曲目。

```
.pause()
```

暂停当前曲目。

```
.play( [name] )
```

开始播放当前的曲目。

新的 1.19

使用可选的 name 参数，直接跳转到和发挥命名的轨道。名称可以指定当调用.add() 。

```
.stop()
```

停止目前的轨道。一样.pause()，但另外的“倒带”轨道开始。

SoundManager

模块中的定义 impact.sound，从 ig.Class 继承

描述

声音管理者需要加载的声音，为他们提供的护理 ig.Music 和 ig.Sound 实例。声音的经理人的一个实例，将自动创建在 ig.soundManager（小写）由 ig.main（ ）函数。

属性

```
.channels
```

此属性已被删除，在 1.17 -使用 ig.Sound.channels，而不是

```
.format
```

声音格式，由目前的浏览器使用。无论是“MP3”或“OGG” 。

浏览器的功能是自动检测和设置相应的格式。当两者都支持，“MP3”格式的青睐，如铬，似乎有更多问题，OGG。

如果你想手动设置，你必须做之前加载任何声音文件。

```
.volume
```

全球体积为从 0.0 到 1.0 的所有 ig.Sound 实例。默认是 1。

谢谢大家支持