Bootstrap包含了三个核心的概念：container、row、col

一、container：

1、margin(left、right)：是为了居中
2、padding(left、right)：
接下来分析的col和container一样，都设置了相同的padding(left、right)，两者是由相同意图的，即所谓nested column。
nested的中文释义即“嵌套”，也就是说栅格系统中的col可能本身也是一个新的栅格系统（分12列）。

- 但是问题来了

按照栅格系统的规定每个col都有padding。如果col想建立新的栅格系统，那col的border就会和实际建立的子栅格系统中间存在padding的空间。
最终导致的结果就如下图：黄色部分是新的子栅格系统，因为col自身存在padding，因此黄色部分就会被像中心挤压（padding-left+padding-right）。

代码如下：

```
<div class="container" style="background:#f2f2f2">
  <div class="col-md-12 bg-warning" style="height:60px">
  </div>
</div>
```

- 那么如何解决?

接下来的row就用来解决这个问题。

3、clearfix：
因为container的通过float实现了responsive，因此clearfix干嘛用的就不多说了。

4、media-query：
这是container另一个重要职责，即根据浏览器resize来动态调整整个container的宽度。

```
.container {
  margin-right: auto;
  margin-left: auto;
  padding-left:  floor(($gutter / 2));
  padding-right: ceil(($gutter / 2));
  @include clearfix;

  @media (min-width: $screen-sm-min) {
    width: $container-sm;
  }
  @media (min-width: $screen-md-min) {
    width: $container-md;
  }
  @media (min-width: $screen-lg-min) {
    width: $container-lg;
  }
}
```

二、row

前面在container提到过“挤压”问题，row的作用正在于此。

抛开clearfix不说，row还设置了margin(left-right)。
它的作用即在于，将row的实际宽度向左右各“拉长”了半个gutter，总共拉长一个gutter。

```
.row {
  margin-left:  ceil(($gutter / -2));
  margin-right: floor(($gutter / -2));
  @include clearfix;
}
```

三、col

这部分代码就多了，col在大类上分为xs（extra small 480px）、sm（small 768px）、md（medium 992px）、lg（large 1200px）。
对每一个大类由细分为 1～12这12个col。

这里只举col-xs-6作为例子：
1、float：left
responsive实现的基础
2、width：50%
这一值是根据 6/12算出来的
3、padding(left、right)：
前面提到过，这就是栅格系统之精髓（也就是gutter的一半）。
4、position：relative，min-height:1px
个人猜测position是为了支持绝对定位，而min-height不太清楚。。
5、回答上面的问题

代码如下，仅仅是在col的周围增加一个row作为包裹。

```
<div class="container" style="background:#f2f2f2">
 <div class="row">
   <div class="col-md-12 bg-warning" style="height:60px">
 </div>
</div>
```

```
.col-xs-6 {
  float: left;

  width: 50%;

  position: relative;
  min-height: 1px;
  padding-left: 15px;
  padding-right: 15px;
}
```

四、总结

如下图，container以及col有一定功能的重合。

1、container根据浏览器的尺寸，将区块设置成一系列预定义的宽度：
xs（extra small 480px）、sm（small 768px）、md（medium 992px）、lg（large 1200px）
因此，container不能嵌套，最多只能互为sibling。并且最好都处于最外层。

2、row是只有一个作用，就是“抵消”掉缓冲空间（padding-left、padding-right）
因此，row紧紧用来做col间的嵌套作用：row包含的内容都是能被分为12部分的。

3、col专注于横向“瓜分”自己parent（一般parent都是row）。

直接根据浏览器resize控制自身宽度
包含内容、并通过padding作为缓冲
包含内容、但是不留缓冲空间（no-padding）
container
√
√

col

√

row

√

五、如何实现布局的reponsive

关键就是mediaquery技术，我们可以看一下grid.scss代码：
相当于一共四种case：
1、默认
2、@media (min-width: $screen-sm-min)
3、@media (min-width: $screen-md-min)
4、@media (min-width: $screen-lg-min)

```
@include make-grid(xs);


// Small grid
//
// Columns, offsets, pushes, and pulls for the small devi
ce range, from phones
// to tablets.

@media (min-width: $screen-sm-min) {
  @include make-grid(sm);
}


// Medium grid
```

```
//
// Columns, offsets, pushes, and pulls for the desktop de
vice range.

@media (min-width: $screen-md-min) {
  @include make-grid(md);
}


// Large grid
//
// Columns, offsets, pushes, and pulls for the large desk
top device range.

@media (min-width: $screen-lg-min) {
  @include make-grid(lg);
}
```

六、reference

http://stackoverflow.com/questions/25723220/why-bootstraps-container-class-has-padding-while-row-class-has-negative-margins

七、关于pull、push、offset

如下图解释，
offset：会造成element位移（向右），同时也会影响sibling节点的位置（也就是说会造成sibling也造成位移）。
push：同样会造成element位移（向右），但是不会影响sibling节点的位置（会发生overlap）。
pull：同样会造成element位移（向左），但是不会影响sibling节点的位置（会发生overlap）。

PS：
要着重注意的时候，一旦我们为一个breakpoint设置了offset、pull、push，那么其他breakpoint如果不需要这三个动作，就要相应设置offset－0，pull－0，push－0。

Since offset uses `margin-left`, and push uses `left`:

- offset will force other columns to move
- push (and pull) will overlap other columns

Here's a visual example: http://www.bootply.com/126557

八、关于breakpoint的reset

reset有何用?
虽然bootstrap设计巧妙，但是受限于其根本原理（基于float），在有些情况会出现布局不受控制。这就需要reset，
reset包括下面四个场景：
1、clearfix
2、pull
3、offset
4、push
一旦我们为一个breakpoint设置了offset、pull、push，那么其他breakpoint如果不需要这三个动作，就要相应设置offset－0，pull－0，push－0。

Responsive column resets

With the four tiers of grids available you're bound to run into issues where, at certain breakpoints, your columns don't clear quite right as one is taller than the other. To fix that, use a combination of a `.clearfix` and our responsive utility classes.

.col-xs-6 .col-sm-3
.col-xs-6 .col-sm-3
.col-xs-6 .col-sm-3
.col-xs-6 .col-sm-3

Copy

```
<div class="row">
```

```
  <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3</div>
  <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3</div>

  <!-- Add the extra clearfix for only the required viewport -->
  <div class="clearfix visible-xs-block"></div>

  <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3</div>
  <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3</div>
</div>
```

In addition to column clearing at responsive breakpoints, you may need to **reset offsets, pushes, or pulls**. See this in action in the grid example.

Copy

```
<div class="row">
  <div class="col-sm-5 col-md-6">.col-sm-5 .col-md-6</div>
  <div class="col-sm-5 col-sm-offset-2 col-md-6 col-md-offset-0">.col-sm-5 .col-sm-
offset-2 .col-md-6 .col-md-offset-0</div>
</div>

<div class="row">
  <div class="col-sm-6 col-md-5 col-lg-6">.col-sm-6 .col-md-5 .col-lg-6</div>
  <div class="col-sm-6 col-md-5 col-md-offset-2 col-lg-6 col-lg-offset-0">.col-sm-6 .col-md-5
.col-md-offset-2 .col-lg-6 .col-lg-offset-0</div>
</div>
```