1.静态属性：

静态属性，本质上也是也是一种属性，只不过他和普通属性不同的是，他是属于类的，即一个类中一个静态属性，只有一份数据；而普通的属性是属于实例的，即每创建一个实例对象，就会有一份数据。

2：静态方法

静态方法属于类方法，而不是实例方法，不需要创建实例对象，可一直接通过类名::静态方法的方式创建，从面向对象的角度来说，当一个方法与实例对象没有逻辑上的关系时，就可以使用静态方法。

3：interface和abstract class

一、 抽象类abstract class

1 ．抽象类是指在 class 前加了 abstract 关键字且存在抽象方法（在类方法 function 关键字前加了 abstract 关键字）的类。

2 ．抽象类不能被直接实例化。抽象类中只定义（或部分实现）子类需要的方法。子类可以通过继承抽象类并通过实现抽象类中的所有抽象方法，使抽象类具体化。

3 ．如果子类需要实例化，前提是它实现了抽象类中的所有抽象方法。如果子类没有全部实现抽象类中的所有抽象方法，那么该子类也是一个抽象类，必须在 class 前面加上 abstract 关键字，并且不能被实例化。

4 ． 如果子类实现了抽象方法，那么 子类中抽象方法的访问控制不能比父类中的抽象方法访问控制更严格，也就是说（A父类，B子类）

　　(1)      如果 A 中 abstract\_func() 声明为 public ，那么 B 中 abstract\_func() 的声明只能是 public ，不能是 protected 或 private

　　(2)      如果 A 中 abstract\_func() 声明为 protected ，那么 B 中 abstract\_func() 的声明可以是 public 或 protected ，但不能是 private

　　(3)      如果 A 中 abstract\_func() 声明为 private ，嘿嘿，不能定义为 private 哦！（ **Fatal error**: Abstract function A::abstract\_func() cannot be declared private ）

二、 接口interface

1 ．抽象类提供了具体实现的标准，而接口则是纯粹的模版。接口只定义功能，而不包含实现的内容。接口用关键字 interface 来声明。

2 ．interface 是完全抽象的，只能声明方法，而且只能声明 public 的方法，不能声明 private 及 protected 的方法，不能定义方法体，也不能声明实例变量 。

3 ．interface 却可以声明常量变量 。但将常量变量放在 interface 中违背了其作为接口的作用而存在的宗旨，也混淆了 interface 与类的不同价值。如果的确需要，可以将其放在相应的 abstract class 或 Class 中。

4 ．任何实现接口的类都要实现接口中所定义的所有方法，否则该类必须声明为 abstract 。

5 ．一个类可以在声明中使用 implements 关键字来实现某个接口。这么做之后，实现接口的具体过程和继承一个仅包含抽象方法的抽象类是一样的。

6 ．一个类可以同时继承一个父类和实现任意多个接口。 extends 子句应该在 implements 子句之前。 PHP 只支持继承自一个父类，因此 extends 关键字后只能跟一个类名。

7 ．接口不可以实现另一个接口，但可以继承多个

三、 抽象类和接口的异同

1. 相同点：

　　(1)      两者都是抽象类，都不能实例化。

　　(2)      interface 实现类及 abstract class 的子类都必须要实现已经声明的抽象方法。

2. 不同点：

　　(1)      interface 需要实现，要用 implements ，而 abstract class 需要继承，要用 extends 。

　　(2)      一个类可以实现多个 interface ，但一个类只能继承一个 abstract class 。

　　(3)      interface 强调特定功能的实现，而 abstract class 强调所属关系。

　　(4)      尽管 interface 实现类及 abstract class 的子类都必须要实现相应的抽象方法，但实现的形式不同。 interface 中的每一个方法都是抽象方法，都只是声明的 (declaration, 没有方法体 ) ，实现类必须要实现。而 abstract class 的子类可以有选择地实现。

　　　　这个选择有两点含义：

　　　　a) abstract class 中并非所有的方法都是抽象的，只有那些冠有 abstract 的方法才是抽象的，子类必须实现。那些没有 abstract 的方法，在 abstract class 中必须定义方法体；

　　　　b) abstract class 的子类在继承它时，对非抽象方法既可以直接继承，也可以覆盖；而对抽象方法，可以选择实现，也可以留给其子类来实现，但此类必须也声明为抽象类。既是抽象类，当然也不能实例化。

　　(5)      abstract class 是 interface 与 class 的中介。 abstract class 在 interface 及 class 中起到了承上启下的作用。

　　　　一方面， abstract class 是抽象的，可以声明抽象方法，以规范子类必须实现的功能；

　　　　另一方面，它又可以定义缺省的方法体，供子类直接使用或覆盖。另外，它还可以定义自己的实例变量，以供子类通过继承来使用。

　　(6)      接口中的抽象方法前不用也不能加 abstract 关键字，默认隐式就是抽象方法，也不能加 final 关键字来防止抽象方法的继承。而抽象类中抽象方法前则必须加上 abstract 表示显示声明为抽象方法。

　　(7)      接口中的抽象方法默认是 public 的，也只能是 public 的，不能用 private ， protected 修饰符修饰。而抽象类中的抽象方法则可以用 public ，protected 来修饰，但不能用 private 。

3. interface 的应用场合

　　(1)      类与类之间需要特定的接口进行协调，而不在乎其如何实现。

　　(2)      作为能够实现特定功能的标识存在，也可以是什么接口方法都没有的纯粹标识。

　　(3)      需要将一组类视为单一的类，而调用者只通过接口来与这组类发生联系。

　　(4)      需要实现特定的多项功能，而这些功能之间可能完全没有任何联系。

4. abstract class 的应用场合

　　一句话，在既需要统一的接口，又需要实例变量或缺省的方法的情况下，就可以使用它。最常见的有：

　　(1)      定义了一组接口，但又不想强迫每个实现类都必须实现所有的接口。可以用 abstract class 定义一组方法体，甚至可以是空方法体，然后由子类选择自己所感兴趣的方法来覆盖。

　　(2)      某些场合下，只靠纯粹的接口不能满足类与类之间的协调，还必需类中表示状态的变量来区别不同的关系。 abstract 的中介作用可以很好地满足这一点。

　　(3)      规范了一组相互协调的方法，其中一些方法是共同的，与状态无关的，可以共享的，无需子类分别实现；而另一些方法却需要各个子类根据自己特定的状态来实现特 定的功能 。