1. Class类
2. base类
3. events

功能：在prototype上添加on、once、off、trigger方法，用于消息的发布订阅

原理：

1. on在this上添加\_\_events={ evtName: [callback1, context1, callback2, context2] } 属性，用于保存订阅的事件
2. once返回this.on(evtName, proxy, context)。当中代理函数在执行once中的callback的同时，this.off(evtName, proxy)
3. trigger(evtName, args) 从this.\_\_events中获取evtName对应的事件数组，并从入参中分离出事件函数参数，循环执行。设置 event.on('all', func)，可作为所有事件的共有方法
4. mixTo(receiver) 将Events上面prototype复制到receiver中

若receiver为函数，则直接对receiver的prototype赋值。（共用原型链可能有问题）

若receiver为Object，则receiver[key]=function(){Events.prototype[key].apply()}

1. aspect

面向切面编程，添加before、after方法

绑定before或after方法：在调用before/after时，通过on绑定事件

包装method方法：通过method.\_\_isAspected判断方法是否被包装过，如果没有，则包装method方法，在原方法前后trigger(‘before:’+method) 和 trigger(‘after:’+method)。

如果在使用的时候有通过before对某个方法添加事件，则在调用该方法时trigger触发before事件。

1. attribute

功能： 统一接口管理prototype.attrs中的属性、prototype对象上的属性、config。

原理流程：

1. 在对象中添加一个this.attrs属性
2. 通过prototype.propsInAttrs记录需要通过attribute统一管理的 prototype对象或prototype.attrs上的属性。
3. mergeInheritedAttrs 循环遍历对象的prototype链
4. 将记录在propsInAttrs中的prototype里有的属性，先复制到prototype.attrs中
5. 然后对proto.attrs中的属性进行序列化处理，将原属性 {key: val} 改为

{ key: { value: val, getter: fn, setter: fn, readOnly: boolean } } 的格式。( getter、setter、readOnly 不属于必须项 )

1. 将序列化后的属性合并到this.attrs中做统一管理
2. mergeUserValue 将config中的属性序列化为 { key: { value: val} } 格式后，合并到this.attrs中。
3. setSetterAttrs 遍历config，对于this.attrs中config对应的属性，如果有setter方法，则调用exports.set 对原config中的数据做初始化处理，（初始化set 时，不会触发change事件，也不会将val的变化保存到this.\_\_changedAttrs中）
4. parseEventsFromAttrs 对this.attrs中的属性名keyName进行正则匹配，如以on/before/after 开头，则通过this.on/before/after( keyName, func ) 或 this.on( ‘change: keyName’, func )监听该方法，并删除原属性。故on/before/after开头的属性，在使用中无法通过this.get() 直接调用。而会在执行keyName方法或set修改keyName属性值的时候，自动触发。
5. copySpecialProps 将this.attrs中propsInAttrs里有记录的属性，通过this.get(key)的方式放到this上。
6. get方法： 获取到this.attrs中对应key属性的value，若该属性中有getter方法，则通过gettter处理获取到的value值并返回。
7. set方法( key, val, options )：options = {silent, override} 判断是否触发change事件，是否覆盖原有属性值（当set的value是Object类型时，需要判断做覆盖或合并处理）

调用格式：set("key", val, options) 或 set({ "key": val, "key2": val2 }, options)

原理：

1. 新建Object对象，赋值为需要修改的键值对 {key: val}。遍历该对象
2. 将set中传入的val作为新的value值，如果this.attrs里key对应的属性中有setter方法，则调用setter方法将返回值作为新的value值。
3. 通过pre = this.get(key) 获取原来的value。 如果pre和val是Object，则判断是否做合并处理
4. 将新的value值赋值给this.attrs[key].value
5. 如果pre与val不同，且不是初始化设置时，判断silent。

如果silent：true ，则将[val, prev] 放到 this.\_\_changedAttrs[key] 中，不触发change事件。在调用this.change() 方法时，再trigger所有在this.\_\_changedAttrs中的属性。

如果silent：false，则触发change事件。

1. change方法：trigger所有在this.\_\_changedAttrs中的属性，对该属性名有添加过ChangeKeyName方法，则调用该方法。