Chapter 0：环境设置

nam, karma, phantomJS, Browserify，jshint(代码检查，lint)

墙内用cnpm代理即可

hello.js 和 hello_spec.js，第一个Karma测试

dollar符号、_符号：对于interpreter来说没有特殊意义

loads.js中的_：provide more consistent cross-environment iteration support for arrays, strings, objects, and arguments objects

==== lodash === 很有用的npm库

https://lodash.com/docs

Part1 中用到了:

.forEach(collection, [iteratee=.identity], [thisArg])

_.indexOf(array, value, [fromIndex=0])

_.rest(array) （数组的第一个元素之后的元素）

等很有用的方法

Part 1 : Scope

Chapter 1: Scopes and Digest

Watching Object: $watch and $digest

$watch:

1, a watch function, specifies the piece of data you’re interested in

Angular中通常是watch expression

2, a listener function, will be called whenever the data changes

$digest:

遍历scope中的所有watchers,

dirty-checking: 检查watch functions对应的值是否真的改变了，如果改变了再调用listener function

过程：调用watch function，比较这次和上次的结果，如果结果不同则这个watcher是dirty的，需要调用listener function

var self=this; 针对Javascript对于this的绑定的处理方法,

```
Scope.prototype.$digest = function() {
var self = this;
_.forEach(this.$$watchers, function(watcher) {
  watcher.watchFn(self);
  watcher.listenerFn();
 });
};
```

此时体现出了:

1，Angular Scope只会遍历所有的watches，不会遍历所有的属性

2，每一个watch function都在$digest的时候被调用。

实现dirty checking: $digest中遍历所有watcher时，比较每个watch function返回值和之前保存的是否不同，function(watcher)中有newValue和oldValue属性。oldValue应该被初始化，而不是undefined.

```
Scope.prototype.$digest = function() { var dirty;
do {
dirty = this.$$digestOnce(); } while (dirty);
};
```

此时体现出Angular 的watch function：

3，idempotent，watch function应该没有副作用，They may be run many times per each digest pass

关于比较的时候基于值还是基于引用？添加一个bool变量,

```
Scope.prototype.$$areEqual = function(newValue, 
oldValue, valueEq){
    if (valueEq){
        return _.isEqual(newValue, oldValue);
    } else{
        return oldValue===newValue || (typeof newValue 
=== 'number' && typeof oldValue === 'number' && 
isNaN(newValue) && isNaN(oldValue));
    }
};
```

如果两个digest在互相watch对方的改变时，会有死循环？设置一个ttl，如果ttl到0了则抛出错误。

```
do{ //Outer loop, run when changes keep occurring
    dirty = this.$digestOnce();
    if (dirty && !(ttl--)){
        throw "10 digest iterations reached";
    }
} while (dirty);
```

性能优化：之前每次digest都是遍历所有watcher，如果遇到一个clean watch是上次的dirty watch，意思是已经遍历过一遍了，并且没有dirty watch，那么当前遍历也没有必要完全进行了，可以减少遍历次数。通过修改dirty布尔值，中止当前digest的遍历。

value-based equal： Angular在做dirty checking的时候默认不做基于值的，而是需要显式设置才会做值的比较。

NaN(Not a number)在JavaScript中总是跟自己不相等的，即NaN=NaN。

$eval：执行参数内的函数

$apply：将外部的库引入到Angular的方法，执行一些Angular不知道的代码。

```
Scope.prototype.$apply = function(expr){
    try{
        return this.$eval(expr);
    }finally{
        this.$digest();
    }
};
```

NaN(Not a number)在JavaScript中总是跟自己不相等的，即NaN=NaN。

deferred execution: AngularJS 自带 $timeout service,

这里实现$evalAsync，一段代码延迟执行（但仍然在当前$digest内执行）

scope内添加一个$$asyncQueue，延迟的代码放在Queue内，然后在$digest中的一次遍历里遍历这个Queue，执行所有task

$applyAsync: 优化连续快速发生的时间，让他们在一次digest中完成。

$$postDigest: 在一个Digest之后运行代码，双美元符号代表是Angular的内部函数。

错误处理：Angular有$errorservice处理错误，这里暂时只打印出来。例如

```
while (this.$$applyAsyncQueue.length){
    try{
        this.$$applyAsyncQueue.shift()();
    } catch (e){
        console.error(e);
    }
}
```

$watchGroup: 一个listener监听多个changes，包装在一个array里。不要实现成一有变化就马上调用listener，而是多次变化之后只调用一次listener

第一章总结:

1. Angular的dirty checking，process包括了 $watch 和 $digest
2. dirty-checking的循环，TTL的机制确保不会无限循环
3. 关于===，基于引用还是基于值?
4. 在digest循环中执行函数的两种方式：$eval,$apply马上执行，$evalAsync,$applyAsync,$$postDigest是稍后执行
5. Exception的处理
6. 销毁watches
7. $watchGroup同时监视多个对象

第二章 Scope Inheritance

Angular的Scope inheritance机制是在Javascript prototypal object inheritance的机制的基础上添加一些内容得到的。

child scope继承parent scope的属性，两者是同一个引用，child scape可以修改parent scape的该属性，也可以watch之。

```
Scope.prototype.$new = function(){
    var ChildScape = function(){};
    ChildScape.prototype = this;
    var child = new ChildScape();
    return child;
};
```

或者

```
Scope.prototype.$new = function(){
    var child = Object.create(this);
    return child;
};
```

（这一段就是Javascript的继承）

shadowing：child scope对parent scope的属性做的修改，只在child及之后的scope chain上被应用。（覆盖）

dot rule: Whenever you use ngModel, there’s got to be a dot in there somewhere. If you don’t have a dot, you’re doing it wrong.

child scope的$$watchers不应该继承parent的，初始化为[]

$digest不应该往chain的上方查找，而是应该向下查找。$$digestOnce需要顺着hierarchy向下遍历。

```
Scope.prototype.$$everyScope = function(fn){
    if (fn(this)){
        return this.$$children.every(function(child){
            return child.$$everyScope(fn);
//recursively invoke fn on child
        });
    }else{
        return false;
    }
};
```

$apply则应该直接从根节点开始遍历

isolated scopes: 部分继承，但是不是全部继承，脱离parent scope的prototype chain。设置Scope.$new(isolated)，isolated=true，构造一个new Scope()，并继承parent Scope的$root，$$asyncQueue, $$postDigestQueue

Substituting the parent scope：$new()函数添加新的参数，选择一个scope作为新scope的parent

```
var parent = parent || this;
```

Destroy a scope：移除所有的watchers，从parent的$$children数组中移除，然后等待Javascript gc。

Summary：

1. child scope的继承，如何创建
2. scope inheritance 和 prototypal inheritance的关系
3. 属性覆盖
4. recursive digestion
5. $digest和$apply的区别
6. Isolated scopes和一般child scope的区别
7. destroy a scope

第三章：Watching Collections

之前的value based $watch，需要保留old value的deep clone，并且需要递归的比较，$watchCollection对此做一些优化

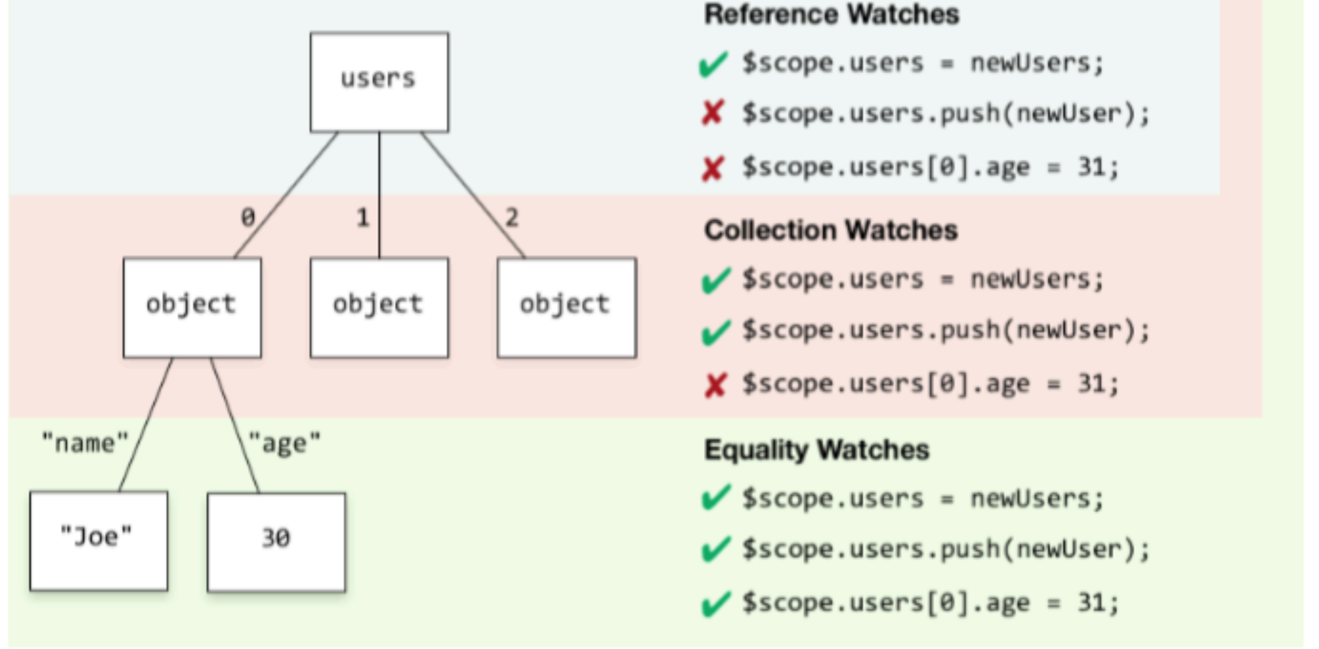

Figure 3.1: The three watch depths in Angular.js. Reference and equality/value-based watches were implemented in Chapter 1. Collection watches are the topic of this chapter.

$watchCollection关注arrays and objects，如果watch function返回non-collection则等同于$watch。

internalWatchFunction应该返回什么呢？重要的只是在连续的调用返回值中有没有做改变，因为对于$watchCollection函数的外部来说怎么改变是不重要的。Angular的做法：对于每个internalWatchFunction，用一个integer counter。

```
var internalWatchFn = function(scope){
    newValue = watchFn(scope);
    //check for changes
    if (newValue != oldValue){
        changeCount+=1;
    }
    oldValue = newValue;
    return changeCount;
};
```

对于NaN，因为NaN!=NaN，所以!=要改成之前实现的 self.$$areEqual。

.isNaN(newValue)&&.isNaN(oldValue)

对于watch的对象是（类）数组，newValue和oldValue先比较数据类型，再比较长度，再逐个遍历比较，这样可以避免每次都要遍历数组以及内部结构

_.isArray只判断是否为数组，需要考虑类似数组的数据结构，如dom node list或者argument list，也当做数组看待

对于watch的对象不是数组，是Object，先比较数据类型，然后遍历所有new object中的属性，和old object作比较，再遍历所有old object的属性，移除new object没有的属性。

这样需要遍历两遍，比较昂贵，优化：记录old object和new object的长度，做长度的比较。

总结:

1. $watchCollection： 我们可以更高效地监视大的array和object，而不用递归比较
2. 如何处理arrays, objects, array-like objects, other values.

Chapter 4: Event System

Angular也是采用publish/subscribe messaging pattern，但是有一点不同：The Angular event system is baked into the scope hierarchy. Rather than having a single point through which all events flow, we have the scope tree where events may propagate up and down. 当publish的时候可以选择向上还是向下，向上是emit（current and ancestor），向下是broadcast（current and descendent）

$on(event name, listener function)

Event object: Angular调用Listeners的机制，同样的event object被传递给每个listener，developer可以添加额外的属性来进行listener之间的通信

var event = {name: eventName}

另外$emit和$broadcast都会在调用完之后返回他们的event object，这样调用者可以在每次调用完之后检查event object的状态。

Angular scope event 有一对属性和DOM event累死，targetScope表示event在哪个scope上出现，currentScope表示listener在哪个scope上。

stopPropagation: 只能在emitted event使用，broadcast不能停止。

preventDefault: Javascript中组织DOM的默认行为，如点击超链接不跳转。Angular中同样，一个布尔值决定是否执行event object的default behavior。

$destroy: destroy a scope (and its children)，this.broadcast(‘$destroy')

Handling exception：listener function发生异常时，不应停止propagating。现在每个emit或者broadcast出现exception时均向外抛出。

总结:

1. publish/subscribe pattern
2. Angular注册listeners，发生事件的机制
3. $emit和$broadcast的区别
4. scope event object
5. stopPropagation, preventDefault, $destroy
6. exception handler