Chapter 9: Expressions And Watches
将Expressions和Scopes结合起来，包括$watch, $watchCollection，$eval（相关的$apply和$evalAsync）
watchFn = parse(watchFn)，之后会变成$parse service

Literal and constant expressions：
returned function除了plain function之外还有一些其他属性，如布尔属性 literal, constant，assign。Literal flag的实现用一个新的函数isLiteral,

```
function isLiteral(ast) {
return ast.body.length === 0 ||    ast.body.length === 1 && (
    ast.body[0].type === AST.Literal ||
    ast.body[0].type === AST.ArrayExpression ||
    ast.body[0].type === AST.ObjectExpression);
}
```

constant flag需要分别考虑每个AST node type来决定如何判断”constantness”，同样用switch ast.type分别考虑。

对Constant expression watching的优化：constant expression总会返回相同的值，所以对应的watch会在第一次被trigger，之后不会再变成dirty了，也就是说这个watch可以被移除了。
实现：$$watchDelegate,

```
if (watchFn.$$watchDelegate) {
return watchFn.$$watchDelegate(self, listenerFn, valueEq, watchFn); }
```

$$表示这是Angular的内部facility，不是用来直接使用的。
parse.js中实现$$watchDelegate,

```
//chapter 9
// a watcher that behaves like any other watcher,
except that it removes itself immediately upon first
invocation
function constantWatchDelegate(scope, listenerFn,
valueEq, watchFn){
    var unwatch = scope.$watch(
        function(){ //wrap in a function with no
$$watchDelegate
            return watchFn(scope);
        },
        function(newValue, oldValue, scope){
            if (_.isFunction(listenerFn)){
                listenerFn.apply(this, arguments);
            }
            unwatch();
        },
        valueEq
    );
    return unwatch;
}
```

One Time expressions：
当watcher的value在这个watcher存在的时候不会改变时，可以用one time expressions，前面加两个引号，如

```
<li ng-repeat="user in users"> {{user.firstName}} {{user.lastName}} </li>
```

对于一个特定user的first name和last name是只读的，不会去修改他，然而Angular还需要在每次digest都去进行dirty check，这可以优化，Angular的优化feature是one-time binding。
one-time watcher和普通watcher的区别是当one-time watcher被使用了之后会自动移除，不存在于之后的digest loop。这个watch被移除的时候是要求这个对象不是undefined的。
如果one time expression是literal的，处理方式稍微有些不同，这里假设expression value是一个collection，然后检查是否包括的所有Items都是defined。

Input tracking：
当一个expression是又多个input expressions组成的，比如a*b由a和b组成，除非组成的input expression变化了，否则就不需要重新检查。实现方法是扩展AST compiler，使得它不仅包括full expression function，也包括一个input expression functions的集合。input tracking是通过维护一个input expressions的数组实现的。每次watch run都是遍历一遍所有inputs看有没有changed。实现markConstantAndWatchExpression方法，在AST内递归调用，使得每个node都有一个toWatch array。
input checking的过程：
1，compiler遍历每个AST节点，根据它的input nodes设置toWatch属性
2，对于每个top-level expression的input生成一个独立的Javascript function body，inputs取决于之前的toWatch属性。
3，compiler的watchFns生成上一步的每个function body的input expression function，与main expression function的inputs属性绑定。
4，当一个expression被watch时，绑定一个inputs watch delegate
5，inputs watch delegate会watch它在inputs找到的每个函数，而不是去watch main expression function

Stageful Filters：
filters一般是pure functions，也就是说filter 通常会对于同样的输入给出同样的输出。然而有时候可能有例外，比如和时间有关的filter，Angular允许对于这种filter设置一个$stateful属性，如果设置为true则不会进行input tracking。

External Assignment：
可能在scope进行外部的assign，这时compiler要生成新的expression function，与main expression function的assign方法进行绑定。assignable AST当且仅当expression只有一个assignment, type是identifier或者member的时候行成。
AST.AssignmentExpression左侧是ast.body[0]，右侧是{type:AST.NGValueParameter}

总结：
本章把Scope和Expression结合起来，并且加入了几个新的特性，对性能优化很重要。

1. Scope如何使用parser对expression进行处理
2. expression如何被标注为constant或者literal
3. watch delegate取代一般的watch，如何对性能进行优化
4. constant watch delegate, one-time binding, input tracking, stageful filters