

# 面向未来的前端类库开发

## — KISSY 类库构想与实践

王保平（玉伯）@ 淘宝

2010/12/18

# Topics

- KISSY 类库介绍
- 种子 seed
- 种子的长大 core
- 基于 mixin 的 UI 组件构建
- 展望



不重复造“相同的”轮子



JavaScript  176  53

An Enjoyable UI Library

Last updated about an hour ago



# KISSY on Github



小巧灵活，简洁实用  
使用起来让人感觉愉悦



## components

- switchable
- calendar
- overlay
- ...

## core

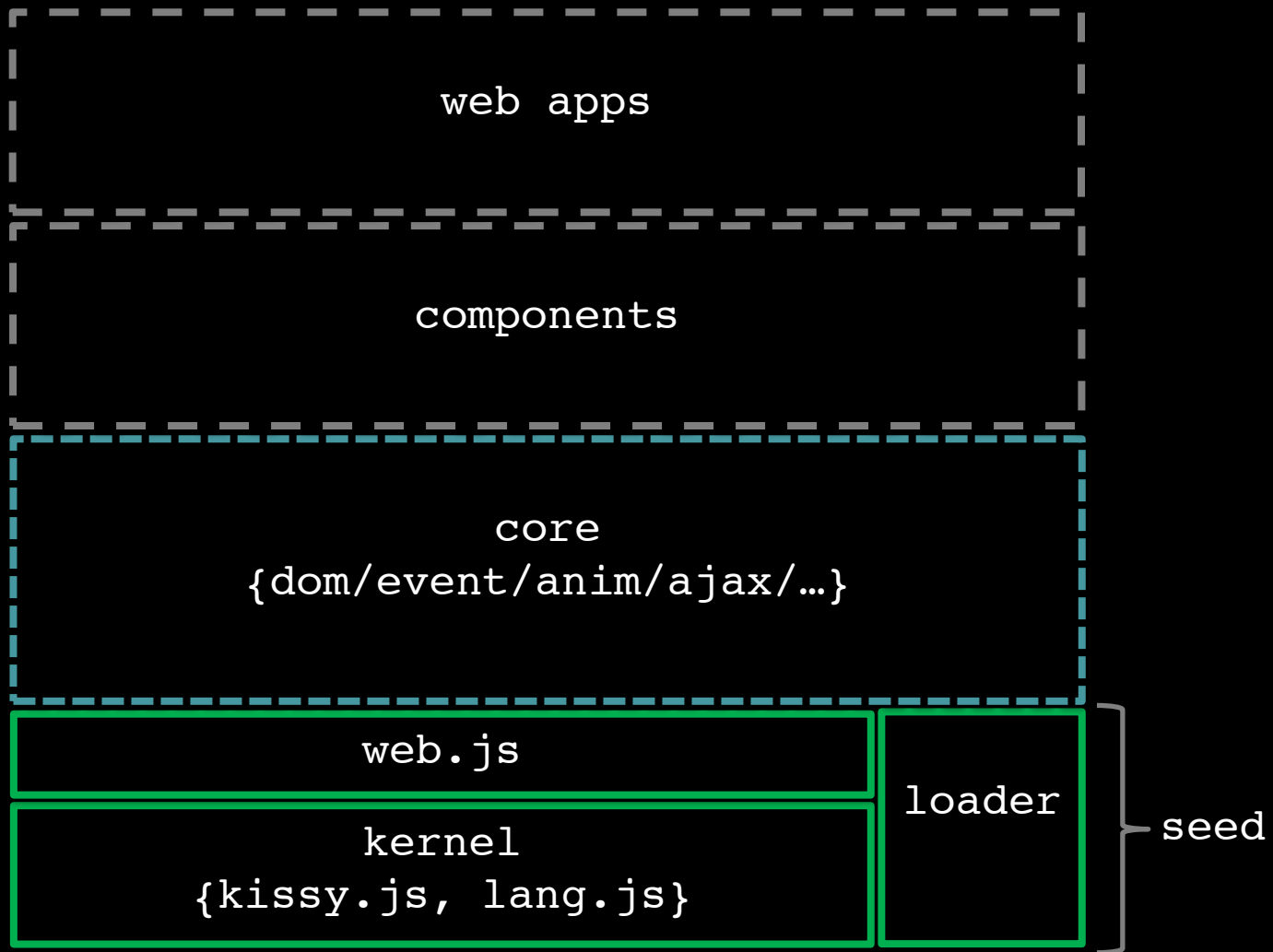
- dom
- event
- ajax
- ...

## seed

- kernel
- web
- loader

种子 seed







# kissy.js

```
(function(host, S) {  
  
    var meta = { mix: function },  
        seed = (host && host[S]) || {};  
  
    host[S] = meta.mix(seed, meta, false);  
  
})(this, 'KISSY');
```

- 原子(meta):

```
var meta = { mix: fn }
```

- 宿主(host):

```
(function(host, S) {  
  })(this, 'KISSY');
```

- 种子(seed):

```
seed = (host && host[S]) || {};  
meta.mix(seed, meta, false);
```

- 种子具有 mix() 和 host 两方面特性。
- 一个系统诞生自一个种子，host 是种子的培育土壤，种子通过不断 mix() 而成长，可生长成任意复杂的系统。
- 传入的种子可以是已存在的复杂系统。

# 用jQuery 做种子

```
<script src="jquery.js"></script>
<script>
    window[ 'KISSY' ] = jQuery;
</script>
<script src="kissy.js"></script>
<script>
    KISSY( ' <p>Hello world!</p>' )
        .appendTo( document.body );
</script>
```

# 种子的长大

```
<script src="labjs.js"></script>
```

```
<script>
```

```
    KISSY.mix(KISSY, $LAB);
```

```
</script>
```

```
<script>
```

```
    KISSY.script('a.js').wait()
```

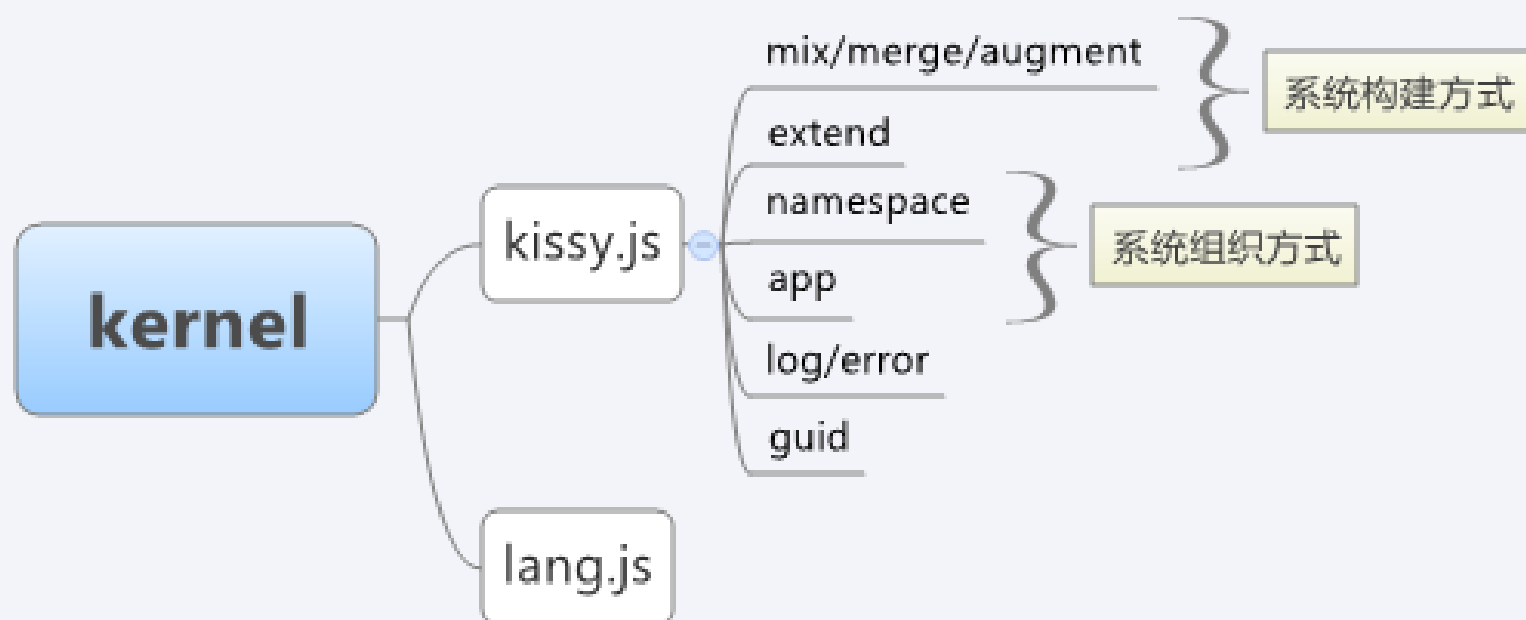
```
        .script('b.js');
```

```
</script>
```

# 基于 seed, 构建基础类库的方式

- ① seed + mix ( 开源类库 )
- ② seed + mix ( 自主研发 )
- ③ seed + mix ( 开源类库 + 自主研发 )

- KISSY 选择第三种方式
- mix 的开源类库有：sizzle, json2
- seed 和其他功能自主研发





# kernel 的特性

- host 无关，可运行在任意宿主环境中

例子一：在 BESENShell 中运行

例子二：在 Photoshop CS5 中运行

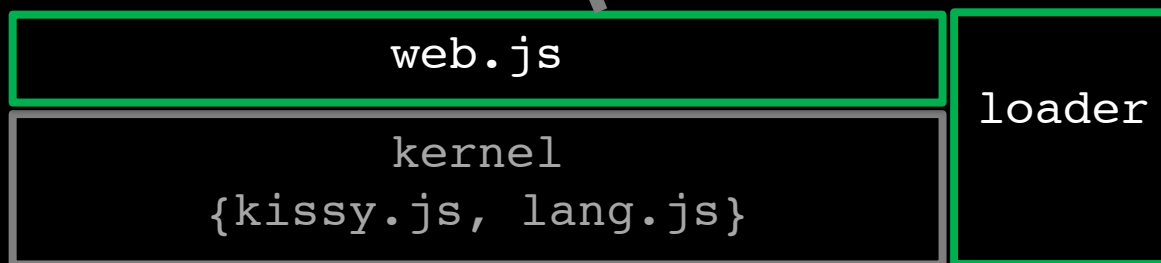
好处一：在服务器端运行

好处二：可装载到任意闭包，沙箱隔离

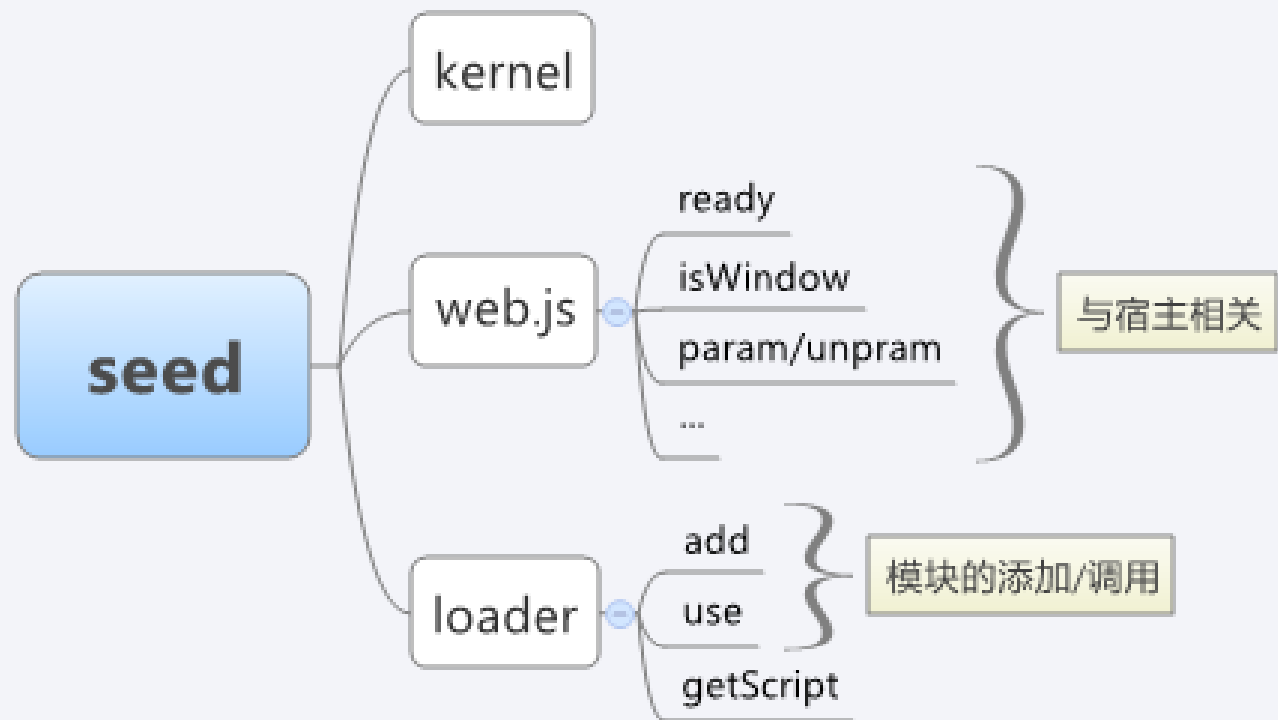
- 与 ES5 兼容

面向未来，春暖花开

与浏览器相关的方法



模块注册/加载/调用



# loader

- 内置模块，无需 add, 直接 use
- 支持静态和动态两种方式

```
KISSY.add( 'modName', {  
    fullpath: 'path/to/mod.js',  
    requires: [ 'core' ]  
});
```

```
KISSY.use( 'modName,calendar', callback);
```

# 用 S.app 构建应用

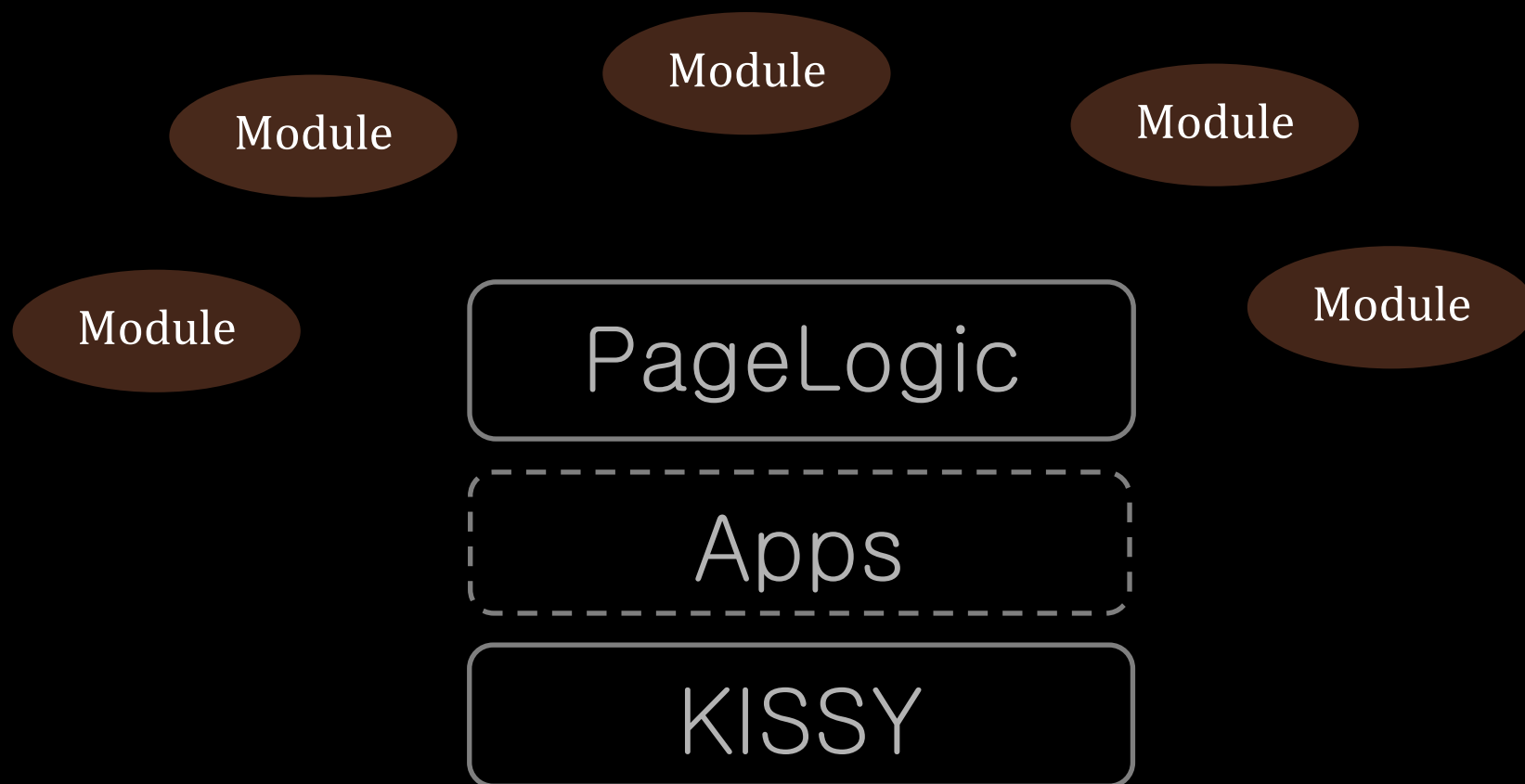
```
KISSY.app( 'D2' );
```

```
D2.add( '克军', {  
    fullpath: '中国/北京/豆瓣'  
});
```

```
D2.use( '克军', function() { ... } );
```

```
D2.namespace( 'China' );
```

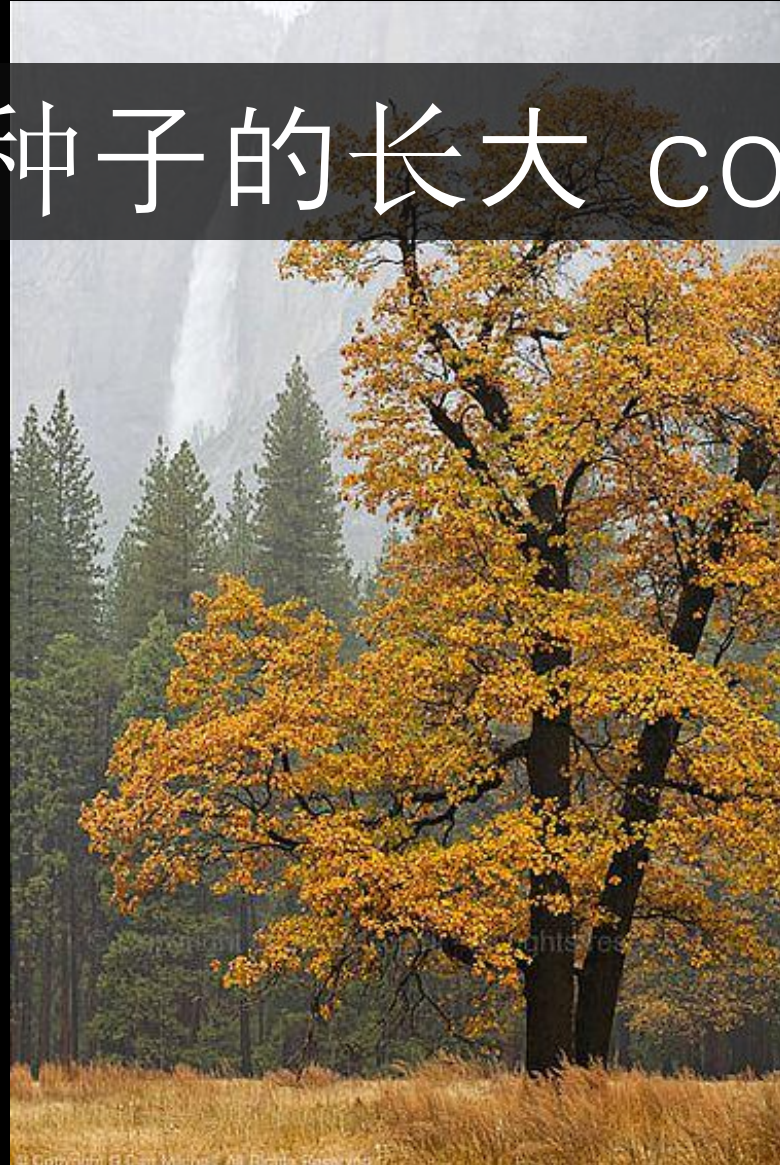
# 前端页面构建



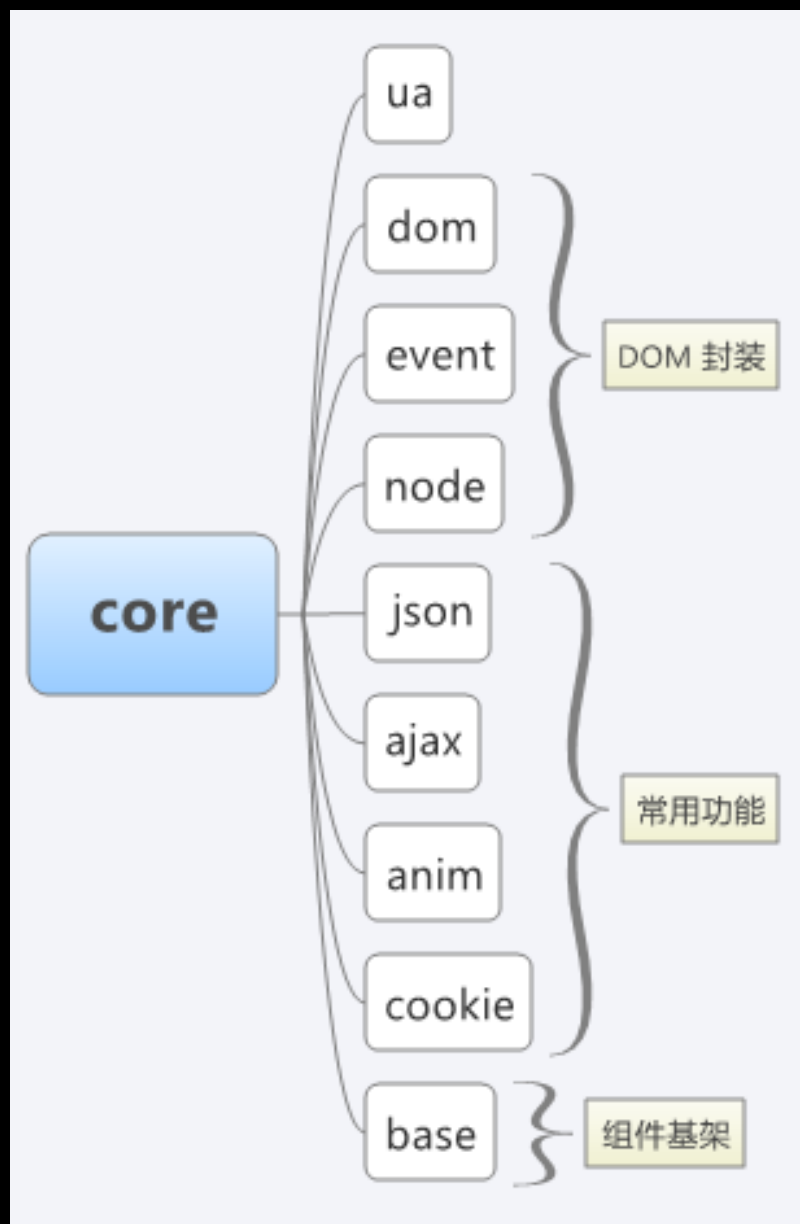
# 小结

- 系统诞生于种子
- 种子通过 mix 长大
- KISSY 选择 seed + mix ( 开源 + 自主研发 )
- seed 里包含 kernel + web.js + loader
- kernel 与 host 无关
- 构建应用的方式 : S.app
- kissy seed 是一个独立的迷你类库

# 种子的长大 core

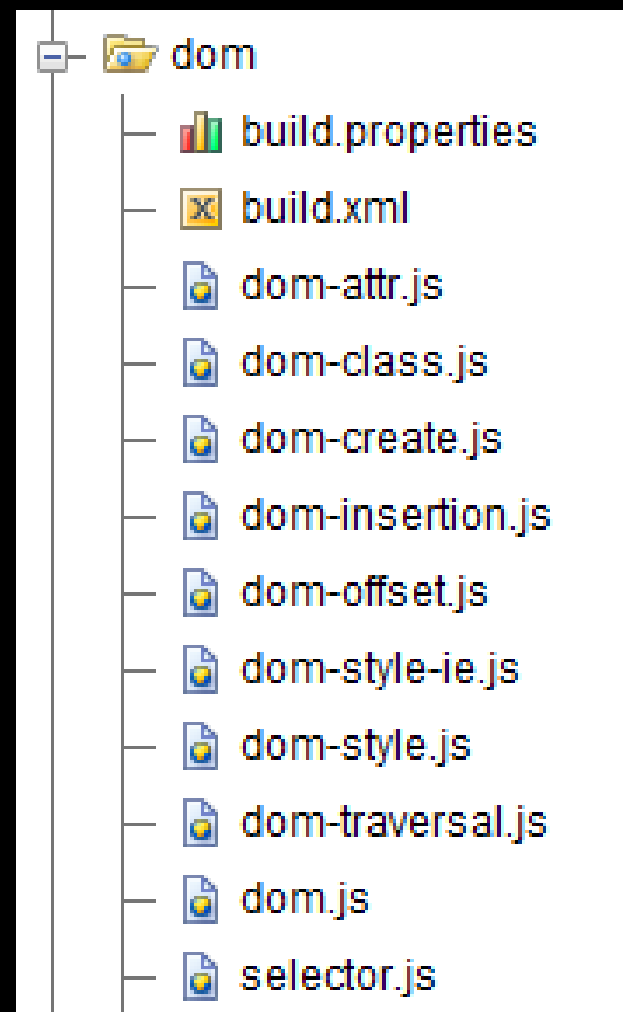






# 模块化、颗粒化

- 模块是 kissy 里文件级的代码组织方式
- 一个模块可以由多个子模块组成



# API 的 20/80 原则

- 实现常用的 20% 功能
  - 满足用户 80% 的需求
- 
- 例子一：selector 默认只支持 #id tag.class
  - 例子二：dom 中无 range 相关方法

core 层的实现方式

# attr 的演化 ( 1 )

```
DOM.getAttribute =  
  function(el, name) {  
  }
```

```
DOM.setAttribute =  
  function(el, name, value) {  
  }
```

# attr 的演化 ( 2 )

```
DOM.attr =  
  function(el, name, value) {  
    if(value === undefined) {  
      return getAttribute(el, name);  
    }  
    setAttribute(el, name, value);  
  }
```

# attr 的演化 ( 3 )

```
DOM.META.getAttribute = fn
```

```
DOM.META.setAttribute = fn
```

```
DOM.attr = publish( 'Attribute' );
```

# attr 的演化 ( 4 )

```
DOM.META = {  
  getAttribute: fn,  
  setAttribute: fn,  
  getStyle: fn,  
  setStyle: fn,  
  ...  
};  
  
S.publish(DOM.META).to(DOM, config);
```



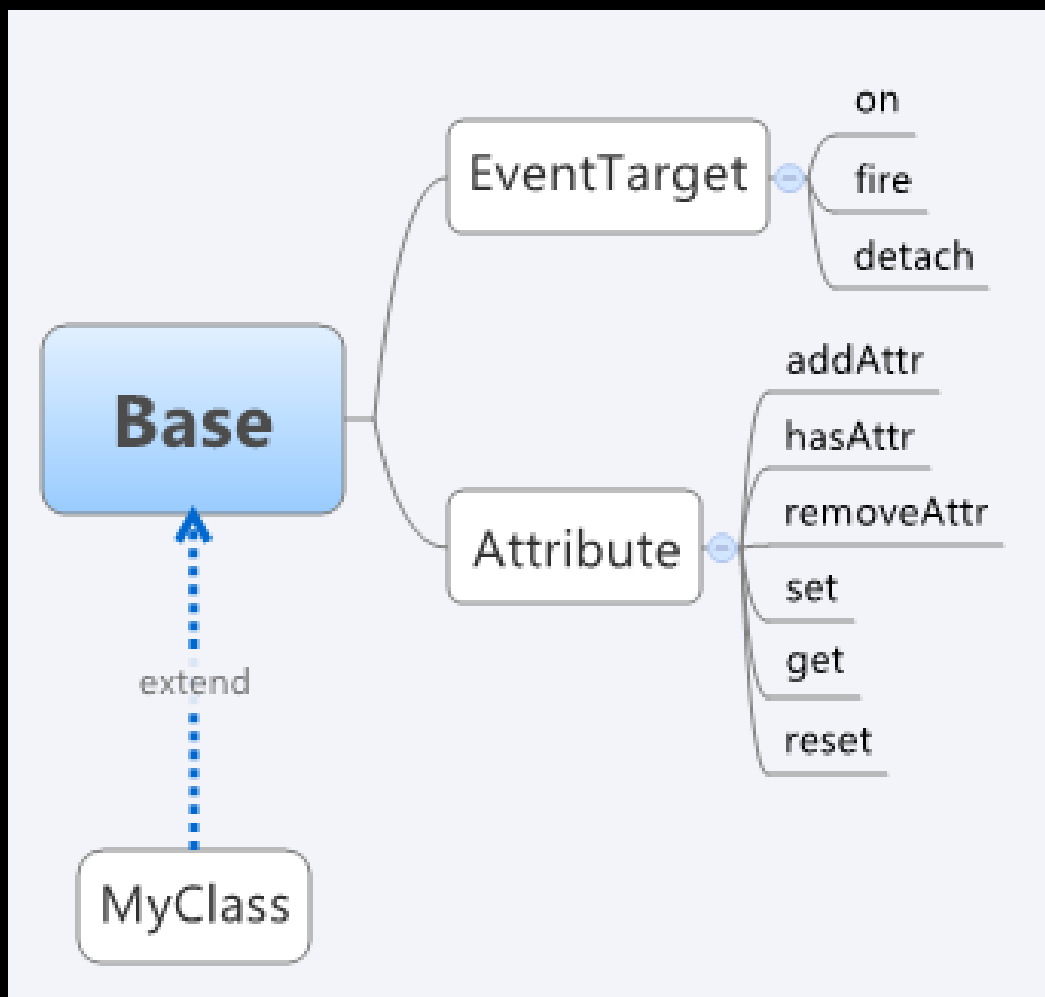
# attr 的演化 ( 5 )

```
S.publish(DOM.META).to(DOM, cfg);  
S.publish(Event.META).to(Event, cfg);  
S.publish([DOM, Event]).to(Node, cfg);  
  
S.Node( '<p>' )  
  .attr( 'data', 'Hedger is an JS ninja!');
```

# 实现方式小结

- 一处只干一件事
  - 用 META 层封装原子方法
  - 用 publish 机制变换为 public api
- 
- 注1：目前尚未完成重构。
  - 注2：要特别感谢 QWrap 团队的分享，publish 构想借鉴了 QWrap 的 retouch 思路。
  - 注3：做好后，会尝试将 META 层独立开源出来，使得能像 sizzle 一样，可复用到其他类库，可自由替换。

# 基于 Base 的组件开发



# Base 示例

```
var S = KISSY;

function Pig(config) {
    Pig.superclass.constructor.call(this, config);
}



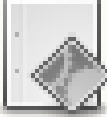

Pig.ATTRS = { name: { value: 'Unknown' } };

S.extend(Pig, S.Base, {
    shout: function() {
        alert('I am ' + this.get('name'));
        this.fire('shout');
    }
});

var pig = new Pig({ name: 'GG Bond' });
pig.on('shout', function() { alert('猪猪侠?'); });
pig.shout();
```

# 小结

- 代码组织方式：模块化、颗粒化
- 代码实现方式：META 层 + publish 变换
- 组件基础构建：Base

	kissy.js	216.6 k
	kissy-min.js	50.4 k
	seed.js	48.2 k
	seed-min.js	11.0 k



# 基于 mixin 的 UI 组件构建

# 组件 components

非 UI 组件

cssreset/ccsgrid/...

datalazyload

flash

...

UI 组件

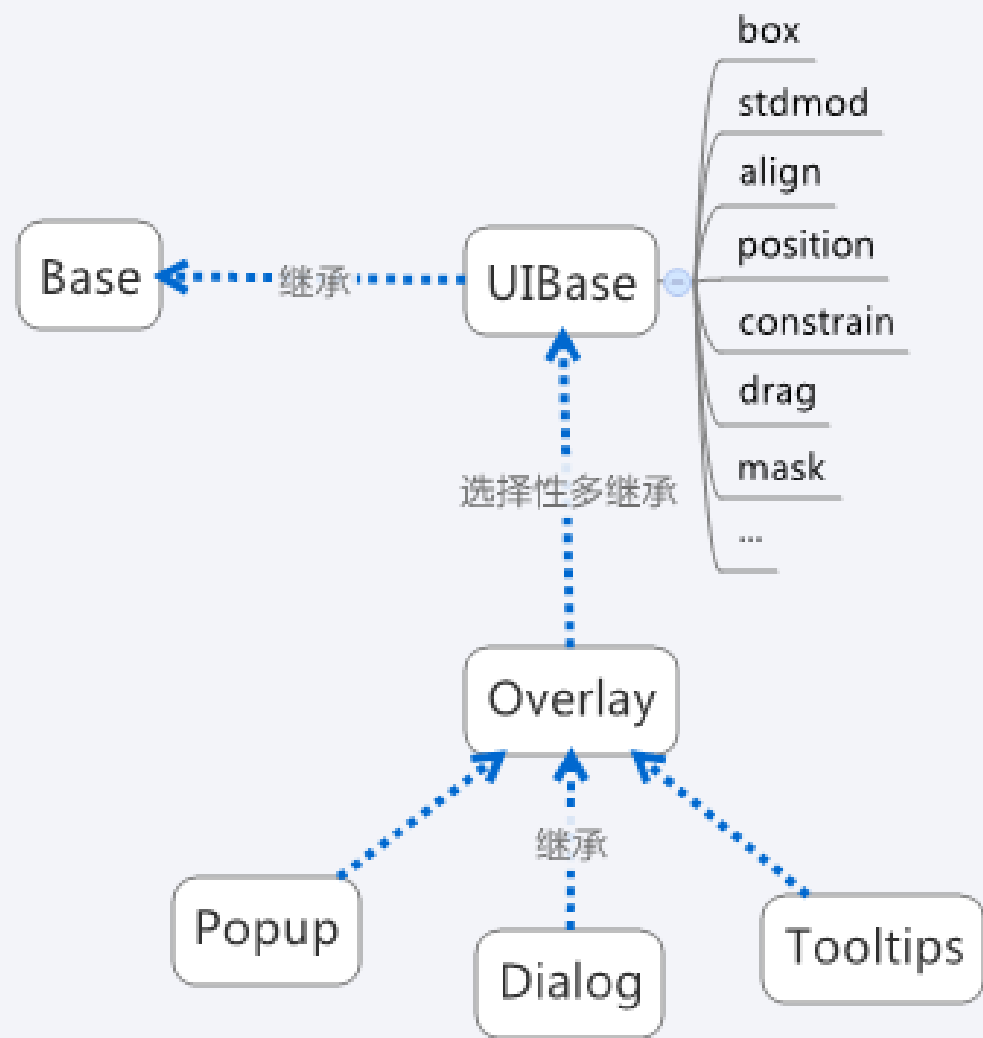
基于 uibase 的组件

overlay

imagezoom

...

其它 ui 组件



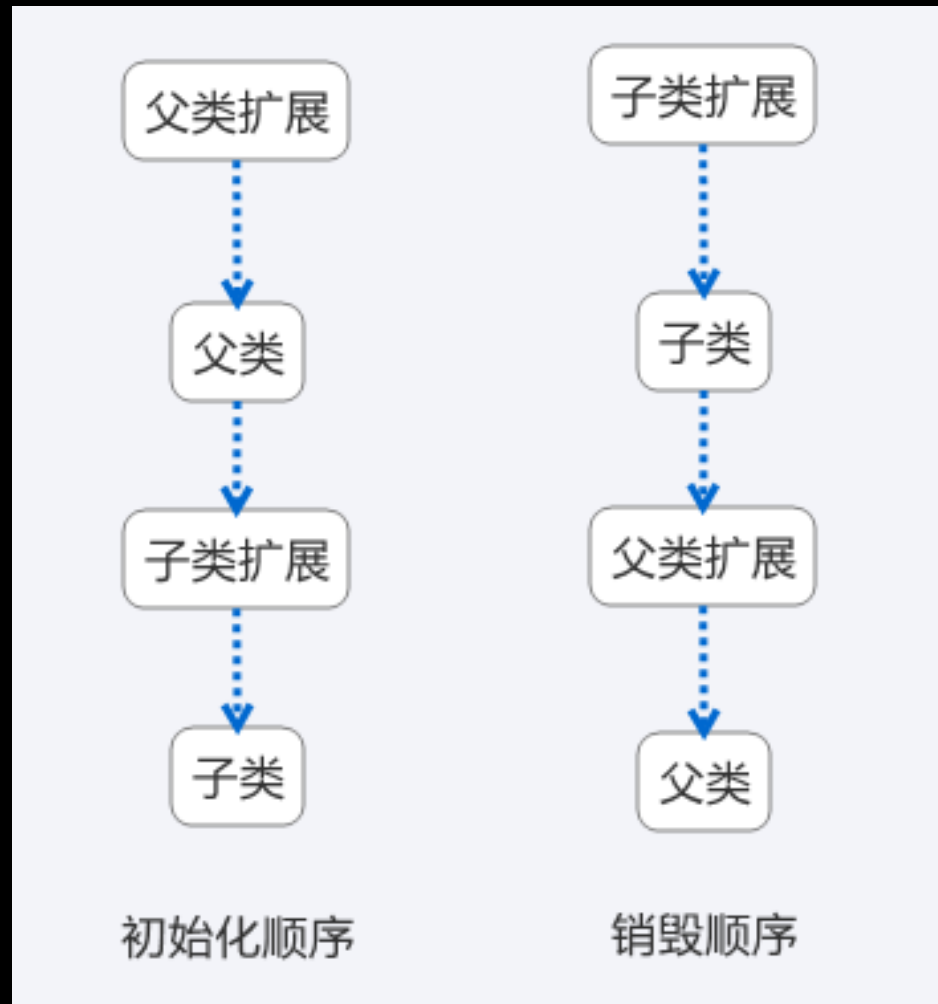


# 基于约定的 UIBase Extension

- constructor / destructor
- UI 的三个阶段：renderUI / bindUI / syncUI
- uiSetAttribute

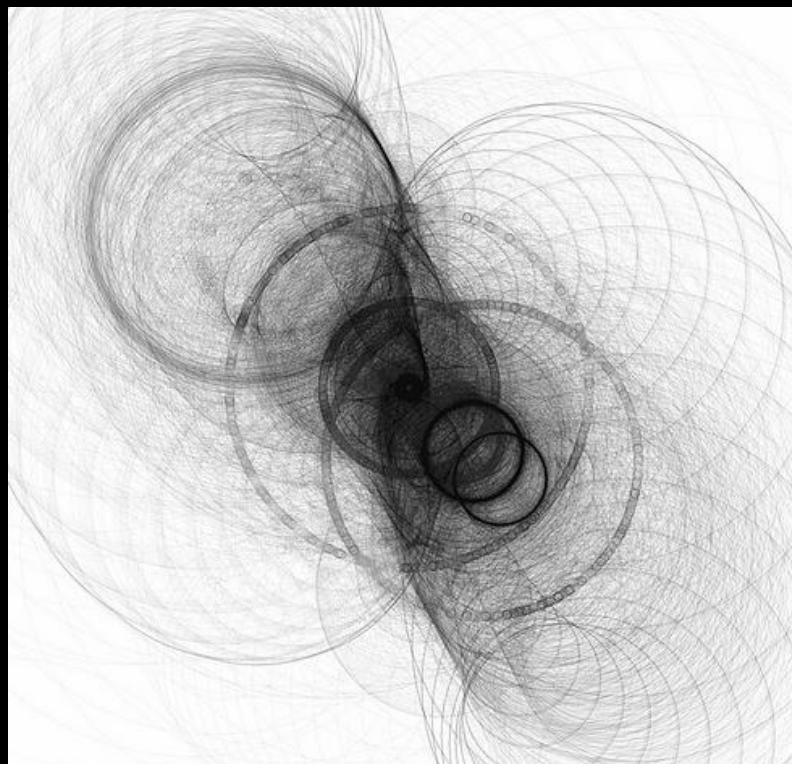
```
function Position() {}  
Position.ATTRS = { x:{}, y:{}, xy:{} }  
Position.prototype = {  
  __renderUI: fn,  
  __bindUI: fn,  
  __syncUI: fn,  
  _uiSetX: fn,  
  __destructor: fn  
}
```

# 模拟 C++ 的多继承



# 基于 mixin 的多继承小结

- 世界很复杂，完美的类抽象很难
  - 适度继承 + 选择性 mix
  - 约定优于配置
- 
- 好处：可复用性高，整体维护方便
  - 不足：成员冲突，门槛稍高



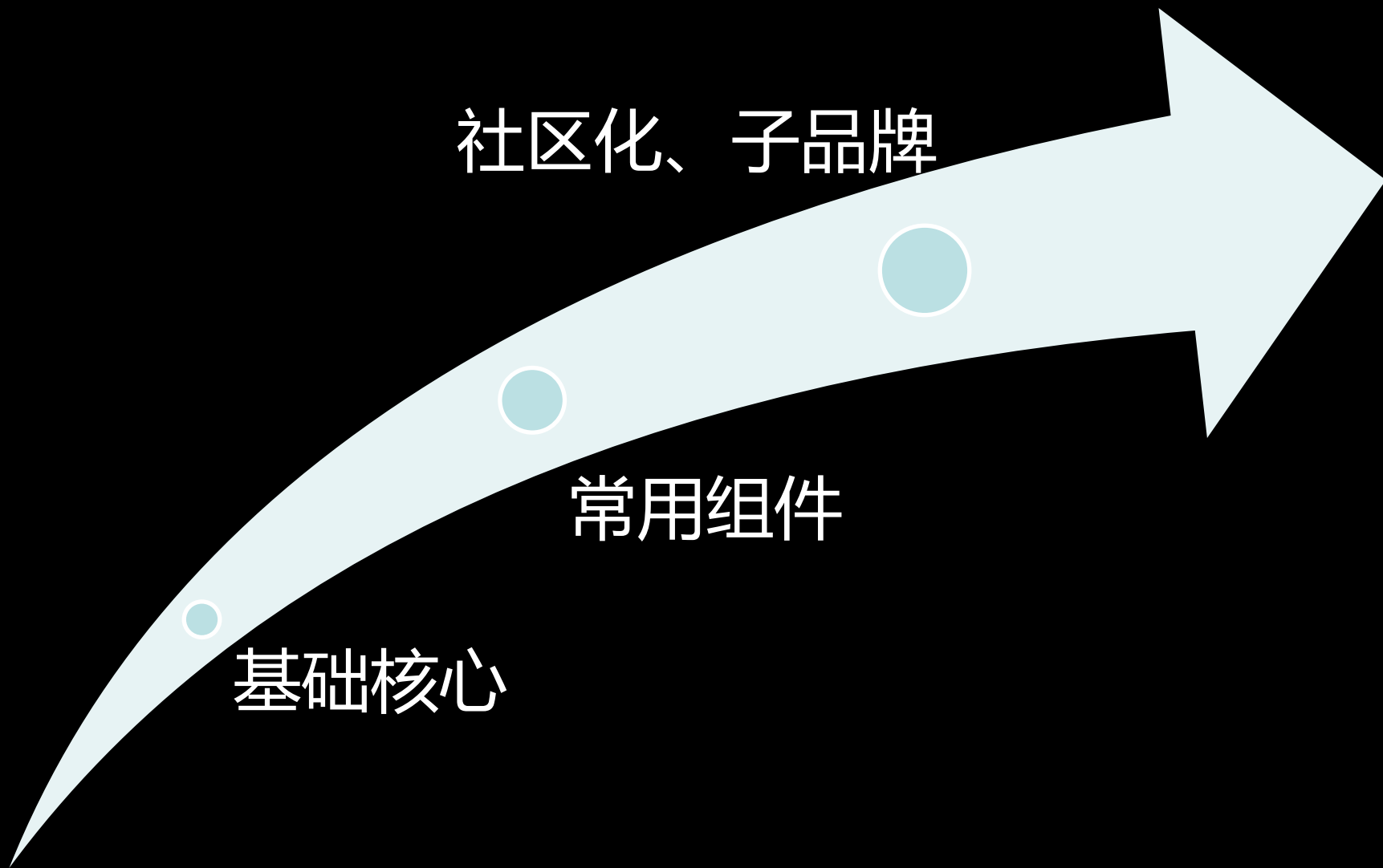
展望

# Roadmap

社区化、子品牌

常用组件

基础核心



## KISSY Team

kissy

KISSY UI Library

kissy-sandbox

实验基地，存放各种成型中的组件

kissy-gallery

展览馆，存放社区贡献的正式组件

kissy-dpl

HTML/CSS 模式库

kissy-editor

KISSY 富文本编辑器

kissy-ajbridge

AJBridge 子品牌区

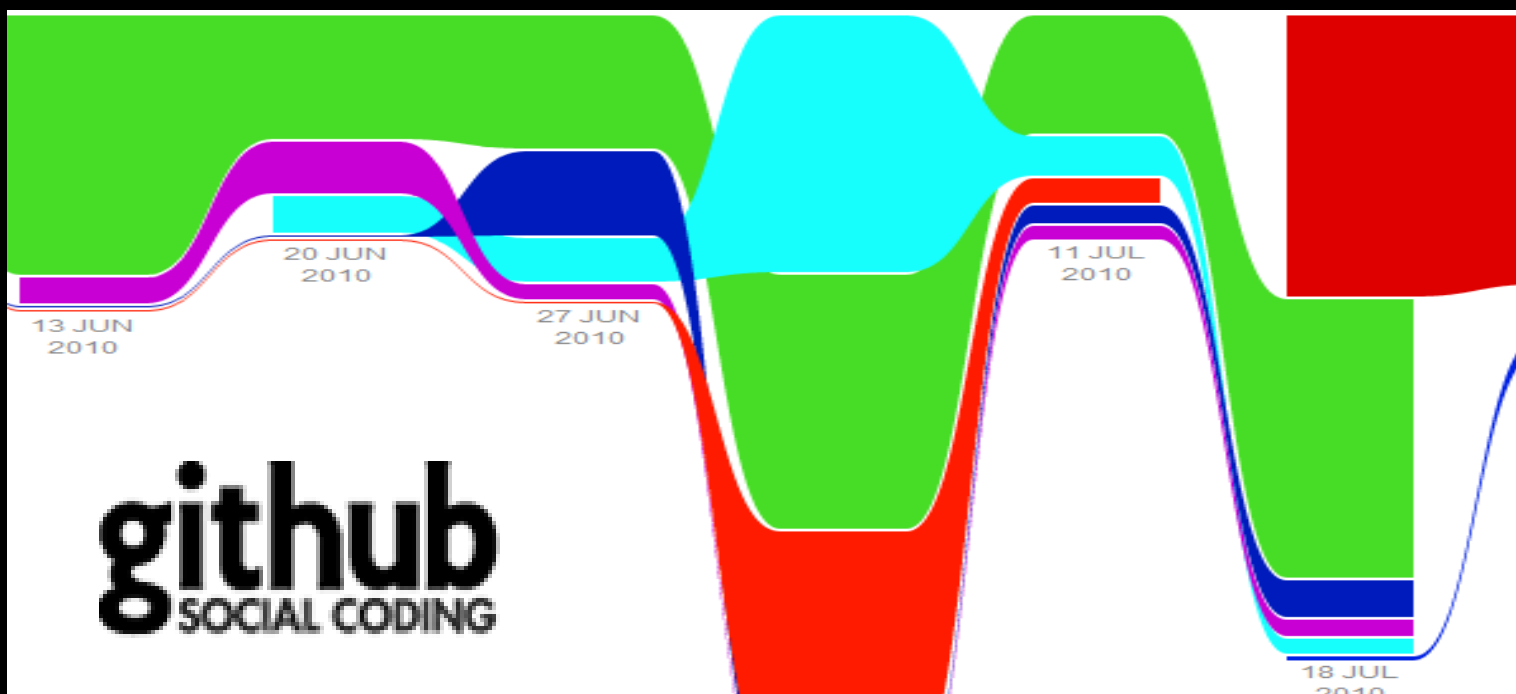
kissy-tools

压缩/打包/文档/测试等自动化工具

kissyteam.github.com

入口 + 文档等资料





期待这里出现你的名字



It' s **NOT** the destination,  
but just where we **BEGIN!**

# Any Questions?

kissy: <http://kissyteam.github.com/>

twitter: @lifesinger

blog: <http://lifesinger.org/>

gtalk: lifesinger@gmail.com

