Adding a Message Component

In this lesson, we will learn how to test two components.

WE'LL COVER THE FOLLOWING

So far, we've learned how to use Shallow Rendering to test a component in isolation, preventing the component sub-tree from rendering.

But in some cases, we want to test components that behave as a group, such as a form label. Keep in mind that this only applies to Presentational Components since they're unaware of app state and logic. In most cases, you'd want to use Shallow Rendering for Container components.

Testing Two Components as a Unit

In the case of the Message and MessageList components, apart from writing their own unit tests, we could want to test them both as a unit as well.

Step 1:

Let's start by creating `components/Message.vue`:

```
/*!
 * Vue.js v2.6.10
 * (c) 2014-2019 Evan You
 * Released under the MIT License.
 */
'use strict';

/*  */

var emptyObject = Object.freeze({});

// These helpers produce better VM code in JS engines due to their
```

```
// These helpers produce better VM code in JS engines due to their
// explicitness and function inlining.
function isUndef (v) {
  return v === undefined || v === null
}

function isDef (v) {
  return v !== undefined && v !== null
}

function isTrue (v) {
  return v === true
}

function isFalse (v) {
  return v === false
}

/**
 * Check if value is primitive.
 */
function isPrimitive (value) {
  return (
    typeof value === 'string' ||
    typeof value === 'number' ||
    // $flow-disable-line
    typeof value === 'symbol' ||
    typeof value === 'boolean'
  )
}

/**
 * Quick object check - this is primarily used to tell
 * Objects from primitive values when we know the value
 * is a JSON-compliant type.
 */
function isObject (obj) {
  return obj !== null && typeof obj === 'object'
}

/**
 * Get the raw type string of a value, e.g., [object Object].
 */
var _toString = Object.prototype.toString;

function toRawType (value) {
  return _toString.call(value).slice(8, -1)
}

/**
 * Strict object type check. Only returns true
 * for plain JavaScript objects.
 */
function isPlainObject (obj) {
  return _toString.call(obj) === '[object Object]'
}

function isRegExp (v) {
  return _toString.call(v) === '[object RegExp]'
}

/**
 * Check if val is a valid array index.
```

```
 */
function isValidArrayIndex (val) {
  var n = parseFloat(String(val));
  return n >= 0 && Math.floor(n) === n && isFinite(val)
}

function isPromise (val) {
  return (
    isDef(val) &&
    typeof val.then === 'function' &&
    typeof val.catch === 'function'
  )
}

/**
 * Convert a value to a string that is actually rendered.
 */
function toString (val) {
  return val == null
    ? ''
    : Array.isArray(val) || (isPlainObject(val) && val.toString === _toString)
      ? JSON.stringify(val, null, 2)
      : String(val)
}

/**
 * Convert an input value to a number for persistence.
 * If the conversion fails, return original string.
 */
function toNumber (val) {
  var n = parseFloat(val);
  return isNaN(n) ? val : n
}

/**
 * Make a map and return a function for checking if a key
 * is in that map.
 */
function makeMap (
  str,
  expectsLowerCase
) {
  var map = Object.create(null);
  var list = str.split(',');
  for (var i = 0; i < list.length; i++) {
    map[list[i]] = true;
  }
  return expectsLowerCase
    ? function (val) { return map[val.toLowerCase()]; }
    : function (val) { return map[val]; }
}

/**
 * Check if a tag is a built-in tag.
 */
var isBuiltInTag = makeMap('slot,component', true);

/**
 * Check if an attribute is a reserved attribute.
 */
var isReservedAttribute = makeMap('key,ref,slot,slot-scope,is');
```

```
/**
 * Remove an item from an array.
 */
function remove (arr, item) {
  if (arr.length) {
    var index = arr.indexOf(item);
    if (index > -1) {
      return arr.splice(index, 1)
    }
  }
}

/**
 * Check whether an object has the property.
 */
var hasOwnProperty = Object.prototype.hasOwnProperty;
function hasOwn (obj, key) {
  return hasOwnProperty.call(obj, key)
}

/**
 * Create a cached version of a pure function.
 */
function cached (fn) {
  var cache = Object.create(null);
  return (function cachedFn (str) {
    var hit = cache[str];
    return hit || (cache[str] = fn(str))
  })
}

/**
 * Camelize a hyphen-delimited string.
 */
var camelizeRE = /-(\w)/g;
var camelize = cached(function (str) {
  return str.replace(camelizeRE, function (_, c) { return c ? c.toUpperCase() : ''; })
});

/**
 * Capitalize a string.
 */
var capitalize = cached(function (str) {
  return str.charAt(0).toUpperCase() + str.slice(1)
});

/**
 * Hyphenate a camelCase string.
 */
var hyphenateRE = /\B([A-Z])/g;
var hyphenate = cached(function (str) {
  return str.replace(hyphenateRE, '-$1').toLowerCase()
});

/**
 * Simple bind polyfill for environments that do not support it,
 * e.g., PhantomJS 1.x. Technically, we don't need this anymore
 * since native bind is now performant enough in most browsers.
 * But removing it would mean breaking code that was able to run in
 * PhantomJS 1.x, so this must be kept for backward compatibility.
 */
```

```
/* istanbul ignore next */
function polyfillBind (fn, ctx) {
  function boundFn (a) {
    var l = arguments.length;
    return l
      ? l > 1
        ? fn.apply(ctx, arguments)
        : fn.call(ctx, a)
      : fn.call(ctx)
  }

  boundFn._length = fn.length;
  return boundFn
}

function nativeBind (fn, ctx) {
  return fn.bind(ctx)
}

var bind = Function.prototype.bind
  ? nativeBind
  : polyfillBind;

/**
 * Convert an Array-like object to a real Array.
 */
function toArray (list, start) {
  start = start || 0;
  var i = list.length - start;
  var ret = new Array(i);
  while (i--) {
    ret[i] = list[i + start];
  }
  return ret
}

/**
 * Mix properties into target object.
 */
function extend (to, _from) {
  for (var key in _from) {
    to[key] = _from[key];
  }
  return to
}

/**
 * Merge an Array of Objects into a single Object.
 */
function toObject (arr) {
  var res = {};
  for (var i = 0; i < arr.length; i++) {
    if (arr[i]) {
      extend(res, arr[i]);
    }
  }
  return res
}

/* eslint-disable no-unused-vars */

/**
```

```
 * Perform no operation.
 * Stubbing args to make Flow happy without leaving useless transpiled code
 * with ...rest (https://flow.org/blog/2017/05/07/Strict-Function-Call-Arity/).
 */
function noop (a, b, c) {}

/**
 * Always return false.
 */
var no = function (a, b, c) { return false; };

/* eslint-enable no-unused-vars */

/**
 * Return the same value.
 */
var identity = function (_) { return _; };

/**
 * Generate a string containing static keys from compiler modules.
 */
function genStaticKeys (modules) {
  return modules.reduce(function (keys, m) {
    return keys.concat(m.staticKeys || [])
  }, []).join(',')
}

/**
 * Check if two values are loosely equal - that is,
 * if they are plain objects, do they have the same shape?
 */
function looseEqual (a, b) {
  if (a === b) { return true }
  var isObjectA = isObject(a);
  var isObjectB = isObject(b);
  if (isObjectA && isObjectB) {
    try {
      var isArrayA = Array.isArray(a);
      var isArrayB = Array.isArray(b);
      if (isArrayA && isArrayB) {
        return a.length === b.length && a.every(function (e, i) {
          return looseEqual(e, b[i])
        })
      } else if (a instanceof Date && b instanceof Date) {
        return a.getTime() === b.getTime()
      } else if (!isArrayA && !isArrayB) {
        var keysA = Object.keys(a);
        var keysB = Object.keys(b);
        return keysA.length === keysB.length && keysA.every(function (key) {
          return looseEqual(a[key], b[key])
        })
      } else {
        /* istanbul ignore next */
        return false
      }
    } catch (e) {
      /* istanbul ignore next */
      return false
    }
  } else if (!isObjectA && !isObjectB) {
    return String(a) === String(b)
  } else {
```

```
    return false
  }
}

/**
 * Return the first index at which a loosely equal value can be
 * found in the array (if value is a plain object, the array must
 * contain an object of the same shape), or -1 if it is not present.
 */
function looseIndexOf (arr, val) {
  for (var i = 0; i < arr.length; i++) {
    if (looseEqual(arr[i], val)) { return i }
  }
  return -1
}

/**
 * Ensure a function is called only once.
 */
function once (fn) {
  var called = false;
  return function () {
    if (!called) {
      called = true;
      fn.apply(this, arguments);
    }
  }
}

var SSR_ATTR = 'data-server-rendered';

var ASSET_TYPES = [
  'component',
  'directive',
  'filter'
];

var LIFECYCLE_HOOKS = [
  'beforeCreate',
  'created',
  'beforeMount',
  'mounted',
  'beforeUpdate',
  'updated',
  'beforeDestroy',
  'destroyed',
  'activated',
  'deactivated',
  'errorCaptured',
  'serverPrefetch'
];

/*  */



var config = ({
  /**
   * Option merge strategies (used in core/util/options)
   */
  // $flow-disable-line
  optionMergeStrategies: Object.create(null),
```

```
  /**
   * Whether to suppress warnings.
   */
  silent: false,

  /**
   * Show production mode tip message on boot?
   */
  productionTip: "development" !== 'production',

  /**
   * Whether to enable devtools
   */
  devtools: "development" !== 'production',

  /**
   * Whether to record perf
   */
  performance: false,

  /**
   * Error handler for watcher errors
   */
  errorHandler: null,

  /**
   * Warn handler for watcher warns
   */
  warnHandler: null,

  /**
   * Ignore certain custom elements
   */
  ignoredElements: [],

  /**
   * Custom user key aliases for v-on
   */
  // $flow-disable-line
  keyCodes: Object.create(null),

  /**
   * Check if a tag is reserved so that it cannot be registered as a
   * component. This is platform-dependent and may be overwritten.
   */
  isReservedTag: no,

  /**
   * Check if an attribute is reserved so that it cannot be used as a component
   * prop. This is platform-dependent and may be overwritten.
   */
  isReservedAttr: no,

  /**
   * Check if a tag is an unknown element.
   * Platform-dependent.
   */
  isUnknownElement: no,

  /**
   * Get the namespace of an element
```

```
   */
  getTagNamespace: noop,

  /**
   * Parse the real tag name for the specific platform.
   */
  parsePlatformTagName: identity,

  /**
   * Check if an attribute must be bound using property, e.g. value
   * Platform-dependent.
   */
  mustUseProp: no,

  /**
   * Perform updates asynchronously. Intended to be used by Vue Test Utils
   * This will significantly reduce performance if set to false.
   */
  async: true,

  /**
   * Exposed for legacy reasons
   */
  _lifecycleHooks: LIFECYCLE_HOOKS
});

/*  */

/**
 * unicode letters used for parsing html tags, component names and property paths.
 * using https://www.w3.org/TR/html53/semantics-scripting.html#potentialcustomelementname
 * skipping \u10000-\uEFFFF due to it freezing up PhantomJS
 */
var unicodeRegExp = /a-zA-Z\u00B7\u00C0-\u00D6\u00D8-\u00F6\u00F8-\u037D\u037F-\u1FFF\u200C-\

/**
 * Check if a string starts with $ or _
 */
function isReserved (str) {
  var c = (str + '').charCodeAt(0);
  return c === 0x24 || c === 0x5F
}

/**
 * Define a property.
 */
function def (obj, key, val, enumerable) {
  Object.defineProperty(obj, key, {
    value: val,
    enumerable: !!enumerable,
    writable: true,
    configurable: true
  });
}

/**
 * Parse simple path.
 */
var bailRE = new RegExp(("[^" + (unicodeRegExp.source) + ".$_\\d]"));
function parsePath (path) {
  if (bailRE.test(path)) {
    return
```

```
  }
  var segments = path.split('.');
  return function (obj) {
    for (var i = 0; i < segments.length; i++) {
      if (!obj) { return }
      obj = obj[segments[i]];
    }
    return obj
  }
}

/*  */

// can we use __proto__?
var hasProto = '__proto__' in {};

// Browser environment sniffing
var inBrowser = typeof window !== 'undefined';
var inWeex = typeof WXEnvironment !== 'undefined' && !!WXEnvironment.platform;
var weexPlatform = inWeex && WXEnvironment.platform.toLowerCase();
var UA = inBrowser && window.navigator.userAgent.toLowerCase();
var isIE = UA && /msie|trident/.test(UA);
var isIE9 = UA && UA.indexOf('msie 9.0') > 0;
var isEdge = UA && UA.indexOf('edge/') > 0;
var isAndroid = (UA && UA.indexOf('android') > 0) || (weexPlatform === 'android');
var isIOS = (UA && /iphone|ipad|ipod|ios/.test(UA)) || (weexPlatform === 'ios');
var isChrome = UA && /chrome\/\d+/.test(UA) && !isEdge;
var isPhantomJS = UA && /phantomjs/.test(UA);
var isFF = UA && UA.match(/firefox\/(\d+)/);

// Firefox has a "watch" function on Object.prototype...
var nativeWatch = ({}).watch;

var supportsPassive = false;
if (inBrowser) {
  try {
    var opts = {};
    Object.defineProperty(opts, 'passive', ({
      get: function get () {
        /* istanbul ignore next */
        supportsPassive = true;
      }
    })); // https://github.com/facebook/flow/issues/285
    window.addEventListener('test-passive', null, opts);
  } catch (e) {}
}

// this needs to be lazy-evaled because vue may be required before
// vue-server-renderer can set VUE_ENV
var _isServer;
var isServerRendering = function () {
  if (_isServer === undefined) {
    /* istanbul ignore if */
    if (!inBrowser && !inWeex && typeof global !== 'undefined') {
      // detect presence of vue-server-renderer and avoid
      // Webpack shimming the process
      _isServer = global['process'] && global['process'].env.VUE_ENV === 'server';
    } else {
      _isServer = false;
    }
  }
  return _isServer
```

```
};

// detect devtools
var devtools = inBrowser && window.__VUE_DEVTOOLS_GLOBAL_HOOK__;

/* istanbul ignore next */
function isNative (Ctor) {
  return typeof Ctor === 'function' && /native code/.test(Ctor.toString())
}

var hasSymbol =
  typeof Symbol !== 'undefined' && isNative(Symbol) &&
  typeof Reflect !== 'undefined' && isNative(Reflect.ownKeys);

var _Set;
/* istanbul ignore if */ // $flow-disable-line
if (typeof Set !== 'undefined' && isNative(Set)) {
  // use native Set when available.
  _Set = Set;
} else {
  // a non-standard Set polyfill that only works with primitive keys.
  _Set = /*@__PURE__*/(function () {
    function Set () {
      this.set = Object.create(null);
    }
    Set.prototype.has = function has (key) {
      return this.set[key] === true
    };
    Set.prototype.add = function add (key) {
      this.set[key] = true;
    };
    Set.prototype.clear = function clear () {
      this.set = Object.create(null);
    };

    return Set;
  }());
}

/*  */

var warn = noop;
var tip = noop;
var generateComponentTrace = (noop); // work around flow check
var formatComponentName = (noop);

{
  var hasConsole = typeof console !== 'undefined';
  var classifyRE = /(?:^|[-_])(\w)/g;
  var classify = function (str) { return str
    .replace(classifyRE, function (c) { return c.toUpperCase(); })
    .replace(/[-_]/g, ''); };

  warn = function (msg, vm) {
    var trace = vm ? generateComponentTrace(vm) : '';

    if (config.warnHandler) {
      config.warnHandler.call(null, msg, vm, trace);
    } else if (hasConsole && (!config.silent)) {
     // console.error(("[Vue warn]: " + msg + trace));
    }
  };
```

```
tip = function (msg, vm) {
  if (hasConsole && (!config.silent)) {
    console.warn("[Vue tip]: " + msg + (
      vm ? generateComponentTrace(vm) : ''
    ));
  }
};

formatComponentName = function (vm, includeFile) {
  if (vm.$root === vm) {
    return '<Root>'
  }
  var options = typeof vm === 'function' && vm.cid != null
    ? vm.options
    : vm._isVue
      ? vm.$options || vm.constructor.options
      : vm;
  var name = options.name || options._componentTag;
  var file = options.__file;
  if (!name && file) {
    var match = file.match(/([^/\\]+)\.vue$/);
    name = match && match[1];
  }

  return (
    (name ? ("<" + (classify(name)) + ">") : "<Anonymous>") +
    (file && includeFile !== false ? (" at " + file) : '')
  )
};

var repeat = function (str, n) {
  var res = '';
  while (n) {
    if (n % 2 === 1) { res += str; }
    if (n > 1) { str += str; }
    n >>= 1;
  }
  return res
};

generateComponentTrace = function (vm) {
  if (vm._isVue && vm.$parent) {
    var tree = [];
    var currentRecursiveSequence = 0;
    while (vm) {
      if (tree.length > 0) {
        var last = tree[tree.length - 1];
        if (last.constructor === vm.constructor) {
          currentRecursiveSequence++;
          vm = vm.$parent;
          continue
        } else if (currentRecursiveSequence > 0) {
          tree[tree.length - 1] = [last, currentRecursiveSequence];
          currentRecursiveSequence = 0;
        }
      }
      tree.push(vm);
      vm = vm.$parent;
    }
    return '\n\nfound in\n\n' + tree
      .map(function (vm, i) { return ("" + (i === 0 ? '---> ' : repeat(' ', 5 + i * 2)) + (
```

```
            ? ((formatComponentName(vm[0])) + "... (" + (vm[1]) + " recursive calls)")
            : formatComponentName(vm))); })
        .join('\n')
    } else {
      return ("\n\n(found in " + (formatComponentName(vm)) + ")")
    }
  };
}

/*  */

var uid = 0;

/**
 * A dep is an observable that can have multiple
 * directives subscribing to it.
 */
var Dep = function Dep () {
  this.id = uid++;
  this.subs = [];
};

Dep.prototype.addSub = function addSub (sub) {
  this.subs.push(sub);
};

Dep.prototype.removeSub = function removeSub (sub) {
  remove(this.subs, sub);
};

Dep.prototype.depend = function depend () {
  if (Dep.target) {
    Dep.target.addDep(this);
  }
};

Dep.prototype.notify = function notify () {
  // stabilize the subscriber list first
  var subs = this.subs.slice();
  if (!config.async) {
    // subs aren't sorted in scheduler if not running async
    // we need to sort them now to make sure they fire in correct
    // order
    subs.sort(function (a, b) { return a.id - b.id; });
  }
  for (var i = 0, l = subs.length; i < l; i++) {
    subs[i].update();
  }
};

// The current target watcher being evaluated.
// This is globally unique because only one watcher
// can be evaluated at a time.
Dep.target = null;
var targetStack = [];

function pushTarget (target) {
  targetStack.push(target);
  Dep.target = target;
}

function popTarget () {
```

```
  targetStack.pop();
  Dep.target = targetStack[targetStack.length - 1];
}

/*  */

var VNode = function VNode (
  tag,
  data,
  children,
  text,
  elm,
  context,
  componentOptions,
  asyncFactory
) {
  this.tag = tag;
  this.data = data;
  this.children = children;
  this.text = text;
  this.elm = elm;
  this.ns = undefined;
  this.context = context;
  this.fnContext = undefined;
  this.fnOptions = undefined;
  this.fnScopeId = undefined;
  this.key = data && data.key;
  this.componentOptions = componentOptions;
  this.componentInstance = undefined;
  this.parent = undefined;
  this.raw = false;
  this.isStatic = false;
  this.isRootInsert = true;
  this.isComment = false;
  this.isCloned = false;
  this.isOnce = false;
  this.asyncFactory = asyncFactory;
  this.asyncMeta = undefined;
  this.isAsyncPlaceholder = false;
};

var prototypeAccessors = { child: { configurable: true } };

// DEPRECATED: alias for componentInstance for backwards compat.
/* istanbul ignore next */
prototypeAccessors.child.get = function () {
  return this.componentInstance
};

Object.defineProperties( VNode.prototype, prototypeAccessors );

var createEmptyVNode = function (text) {
  if ( text === void 0 ) text = '';

  var node = new VNode();
  node.text = text;
  node.isComment = true;
  return node
};

function createTextVNode (val) {
  return new VNode(undefined, undefined, undefined, String(val))
```

```
}

// optimized shallow clone
// used for static nodes and slot nodes because they may be reused across
// multiple renders, cloning them avoids errors when DOM manipulations rely
// on their elm reference.
function cloneVNode (vnode) {
  var cloned = new VNode(
    vnode.tag,
    vnode.data,
    // #7975
    // clone children array to avoid mutating original in case of cloning
    // a child.
    vnode.children && vnode.children.slice(),
    vnode.text,
    vnode.elm,
    vnode.context,
    vnode.componentOptions,
    vnode.asyncFactory
  );
  cloned.ns = vnode.ns;
  cloned.isStatic = vnode.isStatic;
  cloned.key = vnode.key;
  cloned.isComment = vnode.isComment;
  cloned.fnContext = vnode.fnContext;
  cloned.fnOptions = vnode.fnOptions;
  cloned.fnScopeId = vnode.fnScopeId;
  cloned.asyncMeta = vnode.asyncMeta;
  cloned.isCloned = true;
  return cloned
}

/*
 * not type checking this file because flow doesn't play well with
 * dynamically accessing methods on Array prototype
 */

var arrayProto = Array.prototype;
var arrayMethods = Object.create(arrayProto);

var methodsToPatch = [
  'push',
  'pop',
  'shift',
  'unshift',
  'splice',
  'sort',
  'reverse'
];

/**
 * Intercept mutating methods and emit events
 */
methodsToPatch.forEach(function (method) {
  // cache original method
  var original = arrayProto[method];
  def(arrayMethods, method, function mutator () {
    var args = [], len = arguments.length;
    while ( len-- ) args[ len ] = arguments[ len ];

    var result = original.apply(this, args);
    var ob = this.__ob__;
```

```
    var inserted;
    switch (method) {
      case 'push':
      case 'unshift':
        inserted = args;
        break
      case 'splice':
        inserted = args.slice(2);
        break
    }
    if (inserted) { ob.observeArray(inserted); }
    // notify change
    ob.dep.notify();
    return result
  });
});

/*  */

var arrayKeys = Object.getOwnPropertyNames(arrayMethods);

/**
 * In some cases we may want to disable observation inside a component's
 * update computation.
 */
var shouldObserve = true;

function toggleObserving (value) {
  shouldObserve = value;
}

/**
 * Observer class that is attached to each observed
 * object. Once attached, the observer converts the target
 * object's property keys into getter/setters that
 * collect dependencies and dispatch updates.
 */
var Observer = function Observer (value) {
  this.value = value;
  this.dep = new Dep();
  this.vmCount = 0;
  def(value, '__ob__', this);
  if (Array.isArray(value)) {
    if (hasProto) {
      protoAugment(value, arrayMethods);
    } else {
      copyAugment(value, arrayMethods, arrayKeys);
    }
    this.observeArray(value);
  } else {
    this.walk(value);
  }
};

/**
 * Walk through all properties and convert them into
 * getter/setters. This method should only be called when
 * value type is Object.
 */
Observer.prototype.walk = function walk (obj) {
  var keys = Object.keys(obj);
  for (var i = 0; i < keys.length; i++) {
```

```
    defineReactive$$1(obj, keys[i]);
  }
};

/**
 * Observe a list of Array items.
 */
Observer.prototype.observeArray = function observeArray (items) {
  for (var i = 0, l = items.length; i < l; i++) {
    observe(items[i]);
  }
};

// helpers

/**
 * Augment a target Object or Array by intercepting
 * the prototype chain using __proto__
 */
function protoAugment (target, src) {
  /* eslint-disable no-proto */
  target.__proto__ = src;
  /* eslint-enable no-proto */
}

/**
 * Augment a target Object or Array by defining
 * hidden properties.
 */
/* istanbul ignore next */
function copyAugment (target, src, keys) {
  for (var i = 0, l = keys.length; i < l; i++) {
    var key = keys[i];
    def(target, key, src[key]);
  }
}

/**
 * Attempt to create an observer instance for a value,
 * returns the new observer if successfully observed,
 * or the existing observer if the value already has one.
 */
function observe (value, asRootData) {
  if (!isObject(value) || value instanceof VNode) {
    return
  }
  var ob;
  if (hasOwn(value, '__ob__') && value.__ob__ instanceof Observer) {
    ob = value.__ob__;
  } else if (
    shouldObserve &&
    !isServerRendering() &&
    (Array.isArray(value) || isPlainObject(value)) &&
    Object.isExtensible(value) &&
    !value._isVue
  ) {
    ob = new Observer(value);
  }
  if (asRootData && ob) {
    ob.vmCount++;
  }
  return ob
```

```
}

/**
 * Define a reactive property on an Object.
 */
function defineReactive$$1 (
  obj,
  key,
  val,
  customSetter,
  shallow
) {
  var dep = new Dep();

  var property = Object.getOwnPropertyDescriptor(obj, key);
  if (property && property.configurable === false) {
    return
  }

  // cater for pre-defined getter/setters
  var getter = property && property.get;
  var setter = property && property.set;
  if ((!getter || setter) && arguments.length === 2) {
    val = obj[key];
  }

  var childOb = !shallow && observe(val);
  Object.defineProperty(obj, key, {
    enumerable: true,
    configurable: true,
    get: function reactiveGetter () {
      var value = getter ? getter.call(obj) : val;
      if (Dep.target) {
        dep.depend();
        if (childOb) {
          childOb.dep.depend();
          if (Array.isArray(value)) {
            dependArray(value);
          }
        }
      }
      return value
    },
    set: function reactiveSetter (newVal) {
      var value = getter ? getter.call(obj) : val;
      /* eslint-disable no-self-compare */
      if (newVal === value || (newVal !== newVal && value !== value)) {
        return
      }
      /* eslint-enable no-self-compare */
      if (customSetter) {
        customSetter();
      }
      // #7981: for accessor properties without setter
      if (getter && !setter) { return }
      if (setter) {
        setter.call(obj, newVal);
      } else {
        val = newVal;
      }
      childOb = !shallow && observe(newVal);
      dep.notify();
```

```
    }
  });
}

/**
 * Set a property on an object. Adds the new property and
 * triggers change notification if the property doesn't
 * already exist.
 */
function set (target, key, val) {
  if (isUndef(target) || isPrimitive(target)
  ) {
    warn(("Cannot set reactive property on undefined, null, or primitive value: " + ((target)
  }
  if (Array.isArray(target) && isValidArrayIndex(key)) {
    target.length = Math.max(target.length, key);
    target.splice(key, 1, val);
    return val
  }
  if (key in target && !(key in Object.prototype)) {
    target[key] = val;
    return val
  }
  var ob = (target).__ob__;
  if (target._isVue || (ob && ob.vmCount)) {
    warn(
      'Avoid adding reactive properties to a Vue instance or its root $data ' +
      'at runtime - declare it upfront in the data option.'
    );
    return val
  }
  if (!ob) {
    target[key] = val;
    return val
  }
  defineReactive$$1(ob.value, key, val);
  ob.dep.notify();
  return val
}

/**
 * Delete a property and trigger change if necessary.
 */
function del (target, key) {
  if (isUndef(target) || isPrimitive(target)
  ) {
    warn(("Cannot delete reactive property on undefined, null, or primitive value: " + ((targ
  }
  if (Array.isArray(target) && isValidArrayIndex(key)) {
    target.splice(key, 1);
    return
  }
  var ob = (target).__ob__;
  if (target._isVue || (ob && ob.vmCount)) {
    warn(
      'Avoid deleting properties on a Vue instance or its root $data ' +
      '- just set it to null.'
    );
    return
  }
  if (!hasOwn(target, key)) {
    return
```

```
  }
  delete target[key];
  if (!ob) {
    return
  }
  ob.dep.notify();
}

/**
 * Collect dependencies on array elements when the array is touched, since
 * we cannot intercept array element access like property getters.
 */
function dependArray (value) {
  for (var e = (void 0), i = 0, l = value.length; i < l; i++) {
    e = value[i];
    e && e.__ob__ && e.__ob__.dep.depend();
    if (Array.isArray(e)) {
      dependArray(e);
    }
  }
}

/*  */

/**
 * Option overwriting strategies are functions that handle
 * how to merge a parent option value and a child option
 * value into the final value.
 */
var strats = config.optionMergeStrategies;

/**
 * Options with restrictions
 */
{
  strats.el = strats.propsData = function (parent, child, vm, key) {
    if (!vm) {
      warn(
        "option \"" + key + "\" can only be used during instance " +
        'creation with the `new` keyword.'
      );
    }
    return defaultStrat(parent, child)
  };
}

/**
 * Helper that recursively merges two data objects together.
 */
function mergeData (to, from) {
  if (!from) { return to }
  var key, toVal, fromVal;

  var keys = hasSymbol
    ? Reflect.ownKeys(from)
    : Object.keys(from);

  for (var i = 0; i < keys.length; i++) {
    key = keys[i];
    // in case the object is already observed...
    if (key === '__ob__') { continue }
    toVal = to[key];
```

```
    fromVal = from[key];
    if (!hasOwn(to, key)) {
      set(to, key, fromVal);
    } else if (
      toVal !== fromVal &&
      isPlainObject(toVal) &&
      isPlainObject(fromVal)
    ) {
      mergeData(toVal, fromVal);
    }
  }
  return to
}

/**
 * Data
 */
function mergeDataOrFn (
  parentVal,
  childVal,
  vm
) {
  if (!vm) {
    // in a Vue.extend merge, both should be functions
    if (!childVal) {
      return parentVal
    }
    if (!parentVal) {
      return childVal
    }
    // when parentVal & childVal are both present,
    // we need to return a function that returns the
    // merged result of both functions... no need to
    // check if parentVal is a function here because
    // it has to be a function to pass previous merges.
    return function mergedDataFn () {
      return mergeData(
        typeof childVal === 'function' ? childVal.call(this, this) : childVal,
        typeof parentVal === 'function' ? parentVal.call(this, this) : parentVal
      )
    }
  } else {
    return function mergedInstanceDataFn () {
      // instance merge
      var instanceData = typeof childVal === 'function'
        ? childVal.call(vm, vm)
        : childVal;
      var defaultData = typeof parentVal === 'function'
        ? parentVal.call(vm, vm)
        : parentVal;
      if (instanceData) {
        return mergeData(instanceData, defaultData)
      } else {
        return defaultData
      }
    }
  }
}

strats.data = function (
  parentVal,
  childVal,
```

```
  vm
) {
  if (!vm) {
    if (childVal && typeof childVal !== 'function') {
      warn(
        'The "data" option should be a function ' +
        'that returns a per-instance value in component ' +
        'definitions.',
        vm
      );

      return parentVal
    }
    return mergeDataOrFn(parentVal, childVal)
  }

  return mergeDataOrFn(parentVal, childVal, vm)
};

/**
 * Hooks and props are merged as arrays.
 */
function mergeHook (
  parentVal,
  childVal
) {
  var res = childVal
    ? parentVal
      ? parentVal.concat(childVal)
      : Array.isArray(childVal)
        ? childVal
        : [childVal]
    : parentVal;
  return res
    ? dedupeHooks(res)
    : res
}

function dedupeHooks (hooks) {
  var res = [];
  for (var i = 0; i < hooks.length; i++) {
    if (res.indexOf(hooks[i]) === -1) {
      res.push(hooks[i]);
    }
  }
  return res
}

LIFECYCLE_HOOKS.forEach(function (hook) {
  strats[hook] = mergeHook;
});

/**
 * Assets
 *
 * When a vm is present (instance creation), we need to do
 * a three-way merge between constructor options, instance
 * options and parent options.
 */
function mergeAssets (
  parentVal,
  childVal,
```

```
  vm,
  key
) {
  var res = Object.create(parentVal || null);
  if (childVal) {
    assertObjectType(key, childVal, vm);
    return extend(res, childVal)
  } else {
    return res
  }
}

ASSET_TYPES.forEach(function (type) {
  strats[type + 's'] = mergeAssets;
});

/**
 * Watchers.
 *
 * Watchers hashes should not overwrite one
 * another, so we merge them as arrays.
 */
strats.watch = function (
  parentVal,
  childVal,
  vm,
  key
) {
  // work around Firefox's Object.prototype.watch...
  if (parentVal === nativeWatch) { parentVal = undefined; }
  if (childVal === nativeWatch) { childVal = undefined; }
  /* istanbul ignore if */
  if (!childVal) { return Object.create(parentVal || null) }
  {
    assertObjectType(key, childVal, vm);
  }
  if (!parentVal) { return childVal }
  var ret = {};
  extend(ret, parentVal);
  for (var key$1 in childVal) {
    var parent = ret[key$1];
    var child = childVal[key$1];
    if (parent && !Array.isArray(parent)) {
      parent = [parent];
    }
    ret[key$1] = parent
      ? parent.concat(child)
      : Array.isArray(child) ? child : [child];
  }
  return ret
};

/**
 * Other object hashes.
 */
strats.props =
strats.methods =
strats.inject =
strats.computed = function (
  parentVal,
  childVal,
  vm,
```

```
  key
) {
  if (childVal && "development" !== 'production') {
    assertObjectType(key, childVal, vm);
  }
  if (!parentVal) { return childVal }
  var ret = Object.create(null);
  extend(ret, parentVal);
  if (childVal) { extend(ret, childVal); }
  return ret
};
strats.provide = mergeDataOrFn;

/**
 * Default strategy.
 */
var defaultStrat = function (parentVal, childVal) {
  return childVal === undefined
    ? parentVal
    : childVal
};

/**
 * Validate component names
 */
function checkComponents (options) {
  for (var key in options.components) {
    validateComponentName(key);
  }
}

function validateComponentName (name) {
  if (!new RegExp(("^[a-zA-Z][\\-\\.0-9_" + (unicodeRegExp.source) + "]*$")).test(name)) {
    warn(
      'Invalid component name: "' + name + '". Component names ' +
      'should conform to valid custom element name in html5 specification.'
    );
  }
  if (isBuiltInTag(name) || config.isReservedTag(name)) {
    warn(
      'Do not use built-in or reserved HTML elements as component ' +
      'id: ' + name
    );
  }
}

/**
 * Ensure all props option syntax are normalized into the
 * Object-based format.
 */
function normalizeProps (options, vm) {
  var props = options.props;
  if (!props) { return }
  var res = {};
  var i, val, name;
  if (Array.isArray(props)) {
    i = props.length;
    while (i--) {
      val = props[i];
      if (typeof val === 'string') {
        name = camelize(val);
        res[name] = { type: null };
```

```
      } else {
        warn('props must be strings when using array syntax.');
      }
    }
  } else if (isPlainObject(props)) {
    for (var key in props) {
      val = props[key];
      name = camelize(key);
      res[name] = isPlainObject(val)
        ? val
        : { type: val };
    }
  } else {
    warn(
      "Invalid value for option \"props\": expected an Array or an Object, " +
      "but got " + (toRawType(props)) + ".",
      vm
    );
  }
  options.props = res;
}

/**
 * Normalize all injections into Object-based format
 */
function normalizeInject (options, vm) {
  var inject = options.inject;
  if (!inject) { return }
  var normalized = options.inject = {};
  if (Array.isArray(inject)) {
    for (var i = 0; i < inject.length; i++) {
      normalized[inject[i]] = { from: inject[i] };
    }
  } else if (isPlainObject(inject)) {
    for (var key in inject) {
      var val = inject[key];
      normalized[key] = isPlainObject(val)
        ? extend({ from: key }, val)
        : { from: val };
    }
  } else {
    warn(
      "Invalid value for option \"inject\": expected an Array or an Object, " +
      "but got " + (toRawType(inject)) + ".",
      vm
    );
  }
}

/**
 * Normalize raw function directives into object format.
 */
function normalizeDirectives (options) {
  var dirs = options.directives;
  if (dirs) {
    for (var key in dirs) {
      var def$$1 = dirs[key];
      if (typeof def$$1 === 'function') {
        dirs[key] = { bind: def$$1, update: def$$1 };
      }
    }
  }
```

```
}

function assertObjectType (name, value, vm) {
  if (!isPlainObject(value)) {
    warn(
      "Invalid value for option \"" + name + "\": expected an Object, " +
      "but got " + (toRawType(value)) + ".",
      vm
    );
  }
}

/**
 * Merge two option objects into a new one.
 * Core utility used in both instantiation and inheritance.
 */
function mergeOptions (
  parent,
  child,
  vm
) {
  {
    checkComponents(child);
  }

  if (typeof child === 'function') {
    child = child.options;
  }

  normalizeProps(child, vm);
  normalizeInject(child, vm);
  normalizeDirectives(child);

  // Apply extends and mixins on the child options,
  // but only if it is a raw options object that isn't
  // the result of another mergeOptions call.
  // Only merged options has the _base property.
  if (!child._base) {
    if (child.extends) {
      parent = mergeOptions(parent, child.extends, vm);
    }
    if (child.mixins) {
      for (var i = 0, l = child.mixins.length; i < l; i++) {
        parent = mergeOptions(parent, child.mixins[i], vm);
      }
    }
  }

  var options = {};
  var key;
  for (key in parent) {
    mergeField(key);
  }
  for (key in child) {
    if (!hasOwn(parent, key)) {
      mergeField(key);
    }
  }
  function mergeField (key) {
    var strat = strats[key] || defaultStrat;
    options[key] = strat(parent[key], child[key], vm, key);
  }
```

```
  return options
}

/**
 * Resolve an asset.
 * This function is used because child instances need access
 * to assets defined in its ancestor chain.
 */
function resolveAsset (
  options,
  type,
  id,
  warnMissing
) {
  /* istanbul ignore if */
  if (typeof id !== 'string') {
    return
  }
  var assets = options[type];
  // check local registration variations first
  if (hasOwn(assets, id)) { return assets[id] }
  var camelizedId = camelize(id);
  if (hasOwn(assets, camelizedId)) { return assets[camelizedId] }
  var PascalCaseId = capitalize(camelizedId);
  if (hasOwn(assets, PascalCaseId)) { return assets[PascalCaseId] }
  // fallback to prototype chain
  var res = assets[id] || assets[camelizedId] || assets[PascalCaseId];
  if (warnMissing && !res) {
    warn(
      'Failed to resolve ' + type.slice(0, -1) + ': ' + id,
      options
    );
  }
  return res
}

/*  */



function validateProp (
  key,
  propOptions,
  propsData,
  vm
) {
  var prop = propOptions[key];
  var absent = !hasOwn(propsData, key);
  var value = propsData[key];
  // boolean casting
  var booleanIndex = getTypeIndex(Boolean, prop.type);
  if (booleanIndex > -1) {
    if (absent && !hasOwn(prop, 'default')) {
      value = false;
    } else if (value === '' || value === hyphenate(key)) {
      // only cast empty string / same name to boolean if
      // boolean has higher priority
      var stringIndex = getTypeIndex(String, prop.type);
      if (stringIndex < 0 || booleanIndex < stringIndex) {
        value = true;
      }
    }
```

```
  }
  // check default value
  if (value === undefined) {
    value = getPropDefaultValue(vm, prop, key);
    // since the default value is a fresh copy,
    // make sure to observe it.
    var prevShouldObserve = shouldObserve;
    toggleObserving(true);
    observe(value);
    toggleObserving(prevShouldObserve);
  }
  {
    assertProp(prop, key, value, vm, absent);
  }
  return value
}

/**
 * Get the default value of a prop.
 */
function getPropDefaultValue (vm, prop, key) {
  // no default, return undefined
  if (!hasOwn(prop, 'default')) {
    return undefined
  }
  var def = prop.default;
  // warn against non-factory defaults for Object & Array
  if (isObject(def)) {
    warn(
      'Invalid default value for prop "' + key + '": ' +
      'Props with type Object/Array must use a factory function ' +
      'to return the default value.',
      vm
    );
  }
  // the raw prop value was also undefined from previous render,
  // return previous default value to avoid unnecessary watcher trigger
  if (vm && vm.$options.propsData &&
    vm.$options.propsData[key] === undefined &&
    vm._props[key] !== undefined
  ) {
    return vm._props[key]
  }
  // call factory function for non-Function types
  // a value is Function if its prototype is function even across different execution context
  return typeof def === 'function' && getType(prop.type) !== 'Function'
    ? def.call(vm)
    : def
}

/**
 * Assert whether a prop is valid.
 */
function assertProp (
  prop,
  name,
  value,
  vm,
  absent
) {
  if (prop.required && absent) {
    warn(
```

```
      'Missing required prop: "' + name + '"',
      vm
    );
    return
  }
  if (value == null && !prop.required) {
    return
  }
  var type = prop.type;
  var valid = !type || type === true;
  var expectedTypes = [];
  if (type) {
    if (!Array.isArray(type)) {
      type = [type];
    }
    for (var i = 0; i < type.length && !valid; i++) {
      var assertedType = assertType(value, type[i]);
      expectedTypes.push(assertedType.expectedType || '');
      valid = assertedType.valid;
    }
  }

  if (!valid) {
    warn(
      getInvalidTypeMessage(name, value, expectedTypes),
      vm
    );
    return
  }
  var validator = prop.validator;
  if (validator) {
    if (!validator(value)) {
      warn(
        'Invalid prop: custom validator check failed for prop "' + name + '".',
        vm
      );
    }
  }
}

var simpleCheckRE = /^(String|Number|Boolean|Function|Symbol)$/;

function assertType (value, type) {
  var valid;
  var expectedType = getType(type);
  if (simpleCheckRE.test(expectedType)) {
    var t = typeof value;
    valid = t === expectedType.toLowerCase();
    // for primitive wrapper objects
    if (!valid && t === 'object') {
      valid = value instanceof type;
    }
  } else if (expectedType === 'Object') {
    valid = isPlainObject(value);
  } else if (expectedType === 'Array') {
    valid = Array.isArray(value);
  } else {
    valid = value instanceof type;
  }
  return {
    valid: valid,
    expectedType: expectedType
```

```
  }
}

/**
 * Use function string name to check built-in types,
 * because a simple equality check will fail when running
 * across different vms / iframes.
 */
function getType (fn) {
  var match = fn && fn.toString().match(/^\s*function (\w+)/);
  return match ? match[1] : ''
}

function isSameType (a, b) {
  return getType(a) === getType(b)
}

function getTypeIndex (type, expectedTypes) {
  if (!Array.isArray(expectedTypes)) {
    return isSameType(expectedTypes, type) ? 0 : -1
  }
  for (var i = 0, len = expectedTypes.length; i < len; i++) {
    if (isSameType(expectedTypes[i], type)) {
      return i
    }
  }
  return -1
}

function getInvalidTypeMessage (name, value, expectedTypes) {
  var message = "Invalid prop: type check failed for prop \"" + name + "\"." +
    " Expected " + (expectedTypes.map(capitalize).join(', '));
  var expectedType = expectedTypes[0];
  var receivedType = toRawType(value);
  var expectedValue = styleValue(value, expectedType);
  var receivedValue = styleValue(value, receivedType);
  // check if we need to specify expected value
  if (expectedTypes.length === 1 &&
      isExplicable(expectedType) &&
      !isBoolean(expectedType, receivedType)) {
    message += " with value " + expectedValue;
  }
  message += ", got " + receivedType + " ";
  // check if we need to specify received value
  if (isExplicable(receivedType)) {
    message += "with value " + receivedValue + ".";
  }
  return message
}

function styleValue (value, type) {
  if (type === 'String') {
    return ("\"" + value + "\"")
  } else if (type === 'Number') {
    return ("" + (Number(value)))
  } else {
    return ("" + value)
  }
}

function isExplicable (value) {
  var explicitTypes = ['string', 'number', 'boolean'];
```

```
  return explicitTypes.some(function (elem) { return value.toLowerCase() === elem; })
}

function isBoolean () {
  var args = [], len = arguments.length;
  while ( len-- ) args[ len ] = arguments[ len ];

  return args.some(function (elem) { return elem.toLowerCase() === 'boolean'; })
}

/*  */

function handleError (err, vm, info) {
  // Deactivate deps tracking while processing error handler to avoid possible infinite rende
  // See: https://github.com/vuejs/vuex/issues/1505
  pushTarget();
  try {
    if (vm) {
      var cur = vm;
      while ((cur = cur.$parent)) {
        var hooks = cur.$options.errorCaptured;
        if (hooks) {
          for (var i = 0; i < hooks.length; i++) {
            try {
              var capture = hooks[i].call(cur, err, vm, info) === false;
              if (capture) { return }
            } catch (e) {
              globalHandleError(e, cur, 'errorCaptured hook');
            }
          }
        }
      }
    }
    globalHandleError(err, vm, info);
  } finally {
    popTarget();
  }
}

function invokeWithErrorHandling (
  handler,
  context,
  args,
  vm,
  info
) {
  var res;
  try {
    res = args ? handler.apply(context, args) : handler.call(context);
    if (res && !res._isVue && isPromise(res) && !res._handled) {
      res.catch(function (e) { return handleError(e, vm, info + " (Promise/async)"); });
      // issue #9511
      // avoid catch triggering multiple times when nested calls
      res._handled = true;
    }
  } catch (e) {
    handleError(e, vm, info);
  }
  return res
}

function globalHandleError (err, vm, info) {
```

```
  if (config.errorHandler) {
    try {
      return config.errorHandler.call(null, err, vm, info)
    } catch (e) {
      // if the user intentionally throws the original error in the handler,
      // do not log it twice
      if (e !== err) {
        logError(e, null, 'config.errorHandler');
      }
    }
  }
  logError(err, vm, info);
}

function logError (err, vm, info) {
  {
    warn(("Error in " + info + ": \"" + (err.toString()) + "\""), vm);
  }
  /* istanbul ignore else */
  if ((inBrowser || inWeex) && typeof console !== 'undefined') {
    console.error(err);
  } else {
    throw err
  }
}

/*  */

var isUsingMicroTask = false;

var callbacks = [];
var pending = false;

function flushCallbacks () {
  pending = false;
  var copies = callbacks.slice(0);
  callbacks.length = 0;
  for (var i = 0; i < copies.length; i++) {
    copies[i]();
  }
}

// Here we have async deferring wrappers using microtasks.
// In 2.5 we used (macro) tasks (in combination with microtasks).
// However, it has subtle problems when state is changed right before repaint
// (e.g. #6813, out-in transitions).
// Also, using (macro) tasks in event handler would cause some weird behaviors
// that cannot be circumvented (e.g. #7109, #7153, #7546, #7834, #8109).
// So we now use microtasks everywhere, again.
// A major drawback of this tradeoff is that there are some scenarios
// where microtasks have too high a priority and fire in between supposedly
// sequential events (e.g. #4521, #6690, which have workarounds)
// or even between bubbling of the same event (#6566).
var timerFunc;

// The nextTick behavior leverages the microtask queue, which can be accessed
// via either native Promise.then or MutationObserver.
// MutationObserver has wider support, however it is seriously bugged in
// UIWebView in iOS >= 9.3.3 when triggered in touch event handlers. It
// completely stops working after triggering a few times... so, if native
// Promise is available, we will use it:
/* istanbul ignore next, $flow-disable-line */
```

```
if (typeof Promise !== 'undefined' && isNative(Promise)) {
  var p = Promise.resolve();
  timerFunc = function () {
    p.then(flushCallbacks);
    // In problematic UIWebViews, Promise.then doesn't completely break, but
    // it can get stuck in a weird state where callbacks are pushed into the
    // microtask queue but the queue isn't being flushed, until the browser
    // needs to do some other work, e.g. handle a timer. Therefore we can
    // "force" the microtask queue to be flushed by adding an empty timer.
    if (isIOS) { setTimeout(noop); }
  };
  isUsingMicroTask = true;
} else if (!isIE && typeof MutationObserver !== 'undefined' && (
  isNative(MutationObserver) ||
  // PhantomJS and iOS 7.x
  MutationObserver.toString() === '[object MutationObserverConstructor]'
)) {
  // Use MutationObserver where native Promise is not available,
  // e.g. PhantomJS, iOS7, Android 4.4
  // (#6466 MutationObserver is unreliable in IE11)
  var counter = 1;
  var observer = new MutationObserver(flushCallbacks);
  var textNode = document.createTextNode(String(counter));
  observer.observe(textNode, {
    characterData: true
  });
  timerFunc = function () {
    counter = (counter + 1) % 2;
    textNode.data = String(counter);
  };
  isUsingMicroTask = true;
} else if (typeof setImmediate !== 'undefined' && isNative(setImmediate)) {
  // Fallback to setImmediate.
  // Techinically it leverages the (macro) task queue,
  // but it is still a better choice than setTimeout.
  timerFunc = function () {
    setImmediate(flushCallbacks);
  };
} else {
  // Fallback to setTimeout.
  timerFunc = function () {
    setTimeout(flushCallbacks, 0);
  };
}

function nextTick (cb, ctx) {
  var _resolve;
  callbacks.push(function () {
    if (cb) {
      try {
        cb.call(ctx);
      } catch (e) {
        handleError(e, ctx, 'nextTick');
      }
    } else if (_resolve) {
      _resolve(ctx);
    }
  });
  if (!pending) {
    pending = true;
    timerFunc();
  }
```

```
  // $flow-disable-line
  if (!cb && typeof Promise !== 'undefined') {
    return new Promise(function (resolve) {
      _resolve = resolve;
    })
  }
}

/*  */

var mark;
var measure;

{
  var perf = inBrowser && window.performance;
  /* istanbul ignore if */
  if (
    perf &&
    perf.mark &&
    perf.measure &&
    perf.clearMarks &&
    perf.clearMeasures
  ) {
    mark = function (tag) { return perf.mark(tag); };
    measure = function (name, startTag, endTag) {
      perf.measure(name, startTag, endTag);
      perf.clearMarks(startTag);
      perf.clearMarks(endTag);
      // perf.clearMeasures(name)
    };
  }
}

/* not type checking this file because flow doesn't play well with Proxy */

var initProxy;

{
  var allowedGlobals = makeMap(
    'Infinity,undefined,NaN,isFinite,isNaN,' +
    'parseFloat,parseInt,decodeURI,decodeURIComponent,encodeURI,encodeURIComponent,' +
    'Math,Number,Date,Array,Object,Boolean,String,RegExp,Map,Set,JSON,Intl,' +
    'require' // for Webpack/Browserify
  );

  var warnNonPresent = function (target, key) {
    warn(
      "Property or method \"" + key + "\" is not defined on the instance but " +
      'referenced during render. Make sure that this property is reactive, ' +
      'either in the data option, or for class-based components, by ' +
      'initializing the property. ' +
      'See: https://vuejs.org/v2/guide/reactivity.html#Declaring-Reactive-Properties.',
      target
    );
  };

  var warnReservedPrefix = function (target, key) {
    warn(
      "Property \"" + key + "\" must be accessed with \"$data." + key + "\" because " +
      'properties starting with "$" or "_" are not proxied in the Vue instance to ' +
      'prevent conflicts with Vue internals' +
      'See: https://vuejs.org/v2/api/#data',
```

```
    target
  );
};

var hasProxy =
  typeof Proxy !== 'undefined' && isNative(Proxy);

if (hasProxy) {
  var isBuiltInModifier = makeMap('stop,prevent,self,ctrl,shift,alt,meta,exact');
  config.keyCodes = new Proxy(config.keyCodes, {
    set: function set (target, key, value) {
      if (isBuiltInModifier(key)) {
        warn(("Avoid overwriting built-in modifier in config.keyCodes: ." + key));
        return false
      } else {
        target[key] = value;
        return true
      }
    }
  });
}

var hasHandler = {
  has: function has (target, key) {
    var has = key in target;
    var isAllowed = allowedGlobals(key) ||
      (typeof key === 'string' && key.charAt(0) === '_' && !(key in target.$data));
    if (!has && !isAllowed) {
      if (key in target.$data) { warnReservedPrefix(target, key); }
      else { warnNonPresent(target, key); }
    }
    return has || !isAllowed
  }
};

var getHandler = {
  get: function get (target, key) {
    if (typeof key === 'string' && !(key in target)) {
      if (key in target.$data) { warnReservedPrefix(target, key); }
      else { warnNonPresent(target, key); }
    }
    return target[key]
  }
};

initProxy = function initProxy (vm) {
  if (hasProxy) {
    // determine which proxy handler to use
    var options = vm.$options;
    var handlers = options.render && options.render._withStripped
      ? getHandler
      : hasHandler;
    vm._renderProxy = new Proxy(vm, handlers);
  } else {
    vm._renderProxy = vm;
  }
};
}

/*  */

var seenObjects = new _Set();
```

```
/**
 * Recursively traverse an object to evoke all converted
 * getters, so that every nested property inside the object
 * is collected as a "deep" dependency.
 */
function traverse (val) {
  _traverse(val, seenObjects);
  seenObjects.clear();
}

function _traverse (val, seen) {
  var i, keys;
  var isA = Array.isArray(val);
  if ((!isA && !isObject(val)) || Object.isFrozen(val) || val instanceof VNode) {
    return
  }
  if (val.__ob__) {
    var depId = val.__ob__.dep.id;
    if (seen.has(depId)) {
      return
    }
    seen.add(depId);
  }
  if (isA) {
    i = val.length;
    while (i--) { _traverse(val[i], seen); }
  } else {
    keys = Object.keys(val);
    i = keys.length;
    while (i--) { _traverse(val[keys[i]], seen); }
  }
}

/*  */

var normalizeEvent = cached(function (name) {
  var passive = name.charAt(0) === '&';
  name = passive ? name.slice(1) : name;
  var once$$1 = name.charAt(0) === '~'; // Prefixed last, checked first
  name = once$$1 ? name.slice(1) : name;
  var capture = name.charAt(0) === '!';
  name = capture ? name.slice(1) : name;
  return {
    name: name,
    once: once$$1,
    capture: capture,
    passive: passive
  }
});

function createFnInvoker (fns, vm) {
  function invoker () {
    var arguments$1 = arguments;

    var fns = invoker.fns;
    if (Array.isArray(fns)) {
      var cloned = fns.slice();
      for (var i = 0; i < cloned.length; i++) {
        invokeWithErrorHandling(cloned[i], null, arguments$1, vm, "v-on handler");
      }
    } else {
```

```
      // return handler return value for single handlers
      return invokeWithErrorHandling(fns, null, arguments, vm, "v-on handler")
    }
  }
  invoker.fns = fns;
  return invoker
}

function updateListeners (
  on,
  oldOn,
  add,
  remove$$1,
  createOnceHandler,
  vm
) {
  var name, def$$1, cur, old, event;
  for (name in on) {
    def$$1 = cur = on[name];
    old = oldOn[name];
    event = normalizeEvent(name);
    if (isUndef(cur)) {
      warn(
        "Invalid handler for event \"" + (event.name) + "\": got " + String(cur),
        vm
      );
    } else if (isUndef(old)) {
      if (isUndef(cur.fns)) {
        cur = on[name] = createFnInvoker(cur, vm);
      }
      if (isTrue(event.once)) {
        cur = on[name] = createOnceHandler(event.name, cur, event.capture);
      }
      add(event.name, cur, event.capture, event.passive, event.params);
    } else if (cur !== old) {
      old.fns = cur;
      on[name] = old;
    }
  }
  for (name in oldOn) {
    if (isUndef(on[name])) {
      event = normalizeEvent(name);
      remove$$1(event.name, oldOn[name], event.capture);
    }
  }
}

/*  */

function mergeVNodeHook (def, hookKey, hook) {
  if (def instanceof VNode) {
    def = def.data.hook || (def.data.hook = {});
  }
  var invoker;
  var oldHook = def[hookKey];

  function wrappedHook () {
    hook.apply(this, arguments);
    // important: remove merged hook to ensure it's called only once
    // and prevent memory leak
    remove(invoker.fns, wrappedHook);
  }
```

```
  if (isUndef(oldHook)) {
    // no existing hook
    invoker = createFnInvoker([wrappedHook]);
  } else {
    /* istanbul ignore if */
    if (isDef(oldHook.fns) && isTrue(oldHook.merged)) {
      // already a merged invoker
      invoker = oldHook;
      invoker.fns.push(wrappedHook);
    } else {
      // existing plain hook
      invoker = createFnInvoker([oldHook, wrappedHook]);
    }
  }

  invoker.merged = true;
  def[hookKey] = invoker;
}

/*  */

function extractPropsFromVNodeData (
  data,
  Ctor,
  tag
) {
  // we are only extracting raw values here.
  // validation and default values are handled in the child
  // component itself.
  var propOptions = Ctor.options.props;
  if (isUndef(propOptions)) {
    return
  }
  var res = {};
  var attrs = data.attrs;
  var props = data.props;
  if (isDef(attrs) || isDef(props)) {
    for (var key in propOptions) {
      var altKey = hyphenate(key);
      {
        var keyInLowerCase = key.toLowerCase();
        if (
          key !== keyInLowerCase &&
          attrs && hasOwn(attrs, keyInLowerCase)
        ) {
          tip(
            "Prop \"" + keyInLowerCase + "\" is passed to component " +
            (formatComponentName(tag || Ctor)) + ", but the declared prop name is" +
            " \"" + key + "\". " +
            "Note that HTML attributes are case-insensitive and camelCased " +
            "props need to use their kebab-case equivalents when using in-DOM " +
            "templates. You should probably use \"" + altKey + "\" instead of \"" + key + "\"
          );
        }
      }
      checkProp(res, props, key, altKey, true) ||
      checkProp(res, attrs, key, altKey, false);
    }
  }
  return res
}
```

```
function checkProp (
  res,
  hash,
  key,
  altKey,
  preserve
) {
  if (isDef(hash)) {
    if (hasOwn(hash, key)) {
      res[key] = hash[key];
      if (!preserve) {
        delete hash[key];
      }
      return true
    } else if (hasOwn(hash, altKey)) {
      res[key] = hash[altKey];
      if (!preserve) {
        delete hash[altKey];
      }
      return true
    }
  }
  return false
}

/*  */

// The template compiler attempts to minimize the need for normalization by
// statically analyzing the template at compile time.
//
// For plain HTML markup, normalization can be completely skipped because the
// generated render function is guaranteed to return Array<VNode>. There are
// two cases where extra normalization is needed:

// 1. When the children contains components - because a functional component
// may return an Array instead of a single root. In this case, just a simple
// normalization is needed - if any child is an Array, we flatten the whole
// thing with Array.prototype.concat. It is guaranteed to be only 1-level deep
// because functional components already normalize their own children.
function simpleNormalizeChildren (children) {
  for (var i = 0; i < children.length; i++) {
    if (Array.isArray(children[i])) {
      return Array.prototype.concat.apply([], children)
    }
  }
  return children
}

// 2. When the children contains constructs that always generated nested Arrays,
// e.g. <template>, <slot>, v-for, or when the children is provided by user
// with hand-written render functions / JSX. In such cases a full normalization
// is needed to cater to all possible types of children values.
function normalizeChildren (children) {
  return isPrimitive(children)
    ? [createTextVNode(children)]
    : Array.isArray(children)
      ? normalizeArrayChildren(children)
      : undefined
}

function isTextNode (node) {
```

```
  return isDef(node) && isDef(node.text) && isFalse(node.isComment)
}

function normalizeArrayChildren (children, nestedIndex) {
  var res = [];
  var i, c, lastIndex, last;
  for (i = 0; i < children.length; i++) {
    c = children[i];
    if (isUndef(c) || typeof c === 'boolean') { continue }
    lastIndex = res.length - 1;
    last = res[lastIndex];
    //  nested
    if (Array.isArray(c)) {
      if (c.length > 0) {
        c = normalizeArrayChildren(c, ((nestedIndex || '') + "_" + i));
        // merge adjacent text nodes
        if (isTextNode(c[0]) && isTextNode(last)) {
          res[lastIndex] = createTextVNode(last.text + (c[0]).text);
          c.shift();
        }
        res.push.apply(res, c);
      }
    } else if (isPrimitive(c)) {
      if (isTextNode(last)) {
        // merge adjacent text nodes
        // this is necessary for SSR hydration because text nodes are
        // essentially merged when rendered to HTML strings
        res[lastIndex] = createTextVNode(last.text + c);
      } else if (c !== '') {
        // convert primitive to vnode
        res.push(createTextVNode(c));
      }
    } else {
      if (isTextNode(c) && isTextNode(last)) {
        // merge adjacent text nodes
        res[lastIndex] = createTextVNode(last.text + c.text);
      } else {
        // default key for nested array children (likely generated by v-for)
        if (isTrue(children._isVList) &&
          isDef(c.tag) &&
          isUndef(c.key) &&
          isDef(nestedIndex)) {
          c.key = "__vlist" + nestedIndex + "_" + i + "__";
        }
        res.push(c);
      }
    }
  }
  return res
}

/*  */

function initProvide (vm) {
  var provide = vm.$options.provide;
  if (provide) {
    vm._provided = typeof provide === 'function'
      ? provide.call(vm)
      : provide;
  }
}
```

```
function initInjections (vm) {
  var result = resolveInject(vm.$options.inject, vm);
  if (result) {
    toggleObserving(false);
    Object.keys(result).forEach(function (key) {
      /* istanbul ignore else */
      {
        defineReactive$$1(vm, key, result[key], function () {
          warn(
            "Avoid mutating an injected value directly since the changes will be " +
            "overwritten whenever the provided component re-renders. " +
            "injection being mutated: \"" + key + "\"",
            vm
          );
        });
      }
    });
    toggleObserving(true);
  }
}

function resolveInject (inject, vm) {
  if (inject) {
    // inject is :any because flow is not smart enough to figure out cached
    var result = Object.create(null);
    var keys = hasSymbol
      ? Reflect.ownKeys(inject)
      : Object.keys(inject);

    for (var i = 0; i < keys.length; i++) {
      var key = keys[i];
      // #6574 in case the inject object is observed...
      if (key === '__ob__') { continue }
      var provideKey = inject[key].from;
      var source = vm;
      while (source) {
        if (source._provided && hasOwn(source._provided, provideKey)) {
          result[key] = source._provided[provideKey];
          break
        }
        source = source.$parent;
      }
      if (!source) {
        if ('default' in inject[key]) {
          var provideDefault = inject[key].default;
          result[key] = typeof provideDefault === 'function'
            ? provideDefault.call(vm)
            : provideDefault;
        } else {
          warn(("Injection \"" + key + "\" not found"), vm);
        }
      }
    }
    return result
  }
}

/*  */



/**
```

```
 * Runtime helper for resolving raw children VNodes into a slot object.
 */
function resolveSlots (
  children,
  context
) {
  if (!children || !children.length) {
    return {}
  }
  var slots = {};
  for (var i = 0, l = children.length; i < l; i++) {
    var child = children[i];
    var data = child.data;
    // remove slot attribute if the node is resolved as a Vue slot node
    if (data && data.attrs && data.attrs.slot) {
      delete data.attrs.slot;
    }
    // named slots should only be respected if the vnode was rendered in the
    // same context.
    if ((child.context === context || child.fnContext === context) &&
      data && data.slot != null
    ) {
      var name = data.slot;
      var slot = (slots[name] || (slots[name] = []));
      if (child.tag === 'template') {
        slot.push.apply(slot, child.children || []);
      } else {
        slot.push(child);
      }
    } else {
      (slots.default || (slots.default = [])).push(child);
    }
  }
  // ignore slots that contains only whitespace
  for (var name$1 in slots) {
    if (slots[name$1].every(isWhitespace)) {
      delete slots[name$1];
    }
  }
  return slots
}

function isWhitespace (node) {
  return (node.isComment && !node.asyncFactory) || node.text === ' '
}

/*  */

function normalizeScopedSlots (
  slots,
  normalSlots,
  prevSlots
) {
  var res;
  var hasNormalSlots = Object.keys(normalSlots).length > 0;
  var isStable = slots ? !!slots.$stable : !hasNormalSlots;
  var key = slots && slots.$key;
  if (!slots) {
    res = {};
  } else if (slots._normalized) {
    // fast path 1: child component re-render only, parent did not change
    return slots._normalized
```

```
  } else if (
    isStable &&
    prevSlots &&
    prevSlots !== emptyObject &&
    key === prevSlots.$key &&
    !hasNormalSlots &&
    !prevSlots.$hasNormal
  ) {
    // fast path 2: stable scoped slots w/ no normal slots to proxy,
    // only need to normalize once
    return prevSlots
  } else {
    res = {};
    for (var key$1 in slots) {
      if (slots[key$1] && key$1[0] !== '$') {
        res[key$1] = normalizeScopedSlot(normalSlots, key$1, slots[key$1]);
      }
    }
  }
  // expose normal slots on scopedSlots
  for (var key$2 in normalSlots) {
    if (!(key$2 in res)) {
      res[key$2] = proxyNormalSlot(normalSlots, key$2);
    }
  }
  // avoriaz seems to mock a non-extensible $scopedSlots object
  // and when that is passed down this would cause an error
  if (slots && Object.isExtensible(slots)) {
    (slots)._normalized = res;
  }
  def(res, '$stable', isStable);
  def(res, '$key', key);
  def(res, '$hasNormal', hasNormalSlots);
  return res
}

function normalizeScopedSlot(normalSlots, key, fn) {
  var normalized = function () {
    var res = arguments.length ? fn.apply(null, arguments) : fn({});
    res = res && typeof res === 'object' && !Array.isArray(res)
      ? [res] // single vnode
      : normalizeChildren(res);
    return res && (
      res.length === 0 ||
      (res.length === 1 && res[0].isComment) // #9658
    ) ? undefined
      : res
  };
  // this is a slot using the new v-slot syntax without scope. although it is
  // compiled as a scoped slot, render fn users would expect it to be present
  // on this.$slots because the usage is semantically a normal slot.
  if (fn.proxy) {
    Object.defineProperty(normalSlots, key, {
      get: normalized,
      enumerable: true,
      configurable: true
    });
  }
  return normalized
}

function proxyNormalSlot(slots, key) {
```

```
  return function () { return slots[key]; }
}

/*  */

/**
 * Runtime helper for rendering v-for lists.
 */
function renderList (
  val,
  render
) {
  var ret, i, l, keys, key;
  if (Array.isArray(val) || typeof val === 'string') {
    ret = new Array(val.length);
    for (i = 0, l = val.length; i < l; i++) {
      ret[i] = render(val[i], i);
    }
  } else if (typeof val === 'number') {
    ret = new Array(val);
    for (i = 0; i < val; i++) {
      ret[i] = render(i + 1, i);
    }
  } else if (isObject(val)) {
    if (hasSymbol && val[Symbol.iterator]) {
      ret = [];
      var iterator = val[Symbol.iterator]();
      var result = iterator.next();
      while (!result.done) {
        ret.push(render(result.value, ret.length));
        result = iterator.next();
      }
    } else {
      keys = Object.keys(val);
      ret = new Array(keys.length);
      for (i = 0, l = keys.length; i < l; i++) {
        key = keys[i];
        ret[i] = render(val[key], key, i);
      }
    }
  }
  if (!isDef(ret)) {
    ret = [];
  }
  (ret)._isVList = true;
  return ret
}

/*  */

/**
 * Runtime helper for rendering <slot>
 */
function renderSlot (
  name,
  fallback,
  props,
  bindObject
) {
  var scopedSlotFn = this.$scopedSlots[name];
  var nodes;
  if (scopedSlotFn) { // scoped slot
```

```
    props = props || {};
    if (bindObject) {
      if (!isObject(bindObject)) {
        warn(
          'slot v-bind without argument expects an Object',
          this
        );
      }
      props = extend(extend({}, bindObject), props);
    }
    nodes = scopedSlotFn(props) || fallback;
  } else {
    nodes = this.$slots[name] || fallback;
  }

  var target = props && props.slot;
  if (target) {
    return this.$createElement('template', { slot: target }, nodes)
  } else {
    return nodes
  }
}

/*  */

/**
 * Runtime helper for resolving filters
 */
function resolveFilter (id) {
  return resolveAsset(this.$options, 'filters', id, true) || identity
}

/*  */

function isKeyNotMatch (expect, actual) {
  if (Array.isArray(expect)) {
    return expect.indexOf(actual) === -1
  } else {
    return expect !== actual
  }
}

/**
 * Runtime helper for checking keyCodes from config.
 * exposed as Vue.prototype._k
 * passing in eventKeyName as last argument separately for backwards compat
 */
function checkKeyCodes (
  eventKeyCode,
  key,
  builtInKeyCode,
  eventKeyName,
  builtInKeyName
) {
  var mappedKeyCode = config.keyCodes[key] || builtInKeyCode;
  if (builtInKeyName && eventKeyName && !config.keyCodes[key]) {
    return isKeyNotMatch(builtInKeyName, eventKeyName)
  } else if (mappedKeyCode) {
    return isKeyNotMatch(mappedKeyCode, eventKeyCode)
  } else if (eventKeyName) {
    return hyphenate(eventKeyName) !== key
  }
```

```
}

/*  */

/**
 * Runtime helper for merging v-bind="object" into a VNode's data.
 */
function bindObjectProps (
  data,
  tag,
  value,
  asProp,
  isSync
) {
  if (value) {
    if (!isObject(value)) {
      warn(
        'v-bind without argument expects an Object or Array value',
        this
      );
    } else {
      if (Array.isArray(value)) {
        value = toObject(value);
      }
      var hash;
      var loop = function ( key ) {
        if (
          key === 'class' ||
          key === 'style' ||
          isReservedAttribute(key)
        ) {
          hash = data;
        } else {
          var type = data.attrs && data.attrs.type;
          hash = asProp || config.mustUseProp(tag, type, key)
            ? data.domProps || (data.domProps = {})
            : data.attrs || (data.attrs = {});
        }
        var camelizedKey = camelize(key);
        var hyphenatedKey = hyphenate(key);
        if (!(camelizedKey in hash) && !(hyphenatedKey in hash)) {
          hash[key] = value[key];

          if (isSync) {
            var on = data.on || (data.on = {});
            on[("update:" + key)] = function ($event) {
              value[key] = $event;
            };
          }
        }
      };

      for (var key in value) loop( key );
    }
  }
  return data
}

/*  */

/**
 * Runtime helper for rendering static trees.
```

```
 */
function renderStatic (
  index,
  isInFor
) {
  var cached = this._staticTrees || (this._staticTrees = []);
  var tree = cached[index];
  // if has already-rendered static tree and not inside v-for,
  // we can reuse the same tree.
  if (tree && !isInFor) {
    return tree
  }
  // otherwise, render a fresh tree.
  tree = cached[index] = this.$options.staticRenderFns[index].call(
    this._renderProxy,
    null,
    this // for render fns generated for functional component templates
  );
  markStatic(tree, ("__static__" + index), false);
  return tree
}

/**
 * Runtime helper for v-once.
 * Effectively it means marking the node as static with a unique key.
 */
function markOnce (
  tree,
  index,
  key
) {
  markStatic(tree, ("__once__" + index + (key ? ("_" + key) : "")), true);
  return tree
}

function markStatic (
  tree,
  key,
  isOnce
) {
  if (Array.isArray(tree)) {
    for (var i = 0; i < tree.length; i++) {
      if (tree[i] && typeof tree[i] !== 'string') {
        markStaticNode(tree[i], (key + "_" + i), isOnce);
      }
    }
  } else {
    markStaticNode(tree, key, isOnce);
  }
}

function markStaticNode (node, key, isOnce) {
  node.isStatic = true;
  node.key = key;
  node.isOnce = isOnce;
}

/*  */

function bindObjectListeners (data, value) {
  if (value) {
    if (!isPlainObject(value)) {
```

```
      warn(
        'v-on without argument expects an Object value',
        this
      );
    } else {
      var on = data.on = data.on ? extend({}, data.on) : {};
      for (var key in value) {
        var existing = on[key];
        var ours = value[key];
        on[key] = existing ? [].concat(existing, ours) : ours;
      }
    }
  }
  return data
}

/*  */

function resolveScopedSlots (
  fns, // see flow/vnode
  res,
  // the following are added in 2.6
  hasDynamicKeys,
  contentHashKey
) {
  res = res || { $stable: !hasDynamicKeys };
  for (var i = 0; i < fns.length; i++) {
    var slot = fns[i];
    if (Array.isArray(slot)) {
      resolveScopedSlots(slot, res, hasDynamicKeys);
    } else if (slot) {
      // marker for reverse proxying v-slot without scope on this.$slots
      if (slot.proxy) {
        slot.fn.proxy = true;
      }
      res[slot.key] = slot.fn;
    }
  }
  if (contentHashKey) {
    (res).$key = contentHashKey;
  }
  return res
}

/*  */

function bindDynamicKeys (baseObj, values) {
  for (var i = 0; i < values.length; i += 2) {
    var key = values[i];
    if (typeof key === 'string' && key) {
      baseObj[values[i]] = values[i + 1];
    } else if (key !== '' && key !== null) {
      // null is a speical value for explicitly removing a binding
      warn(
        ("Invalid value for dynamic directive argument (expected string or null): " + key),
        this
      );
    }
  }
  return baseObj
}
```

```
// helper to dynamically append modifier runtime markers to event names.
// ensure only append when value is already string, otherwise it will be cast
// to string and cause the type check to miss.
function prependModifier (value, symbol) {
  return typeof value === 'string' ? symbol + value : value
}

/*  */

function installRenderHelpers (target) {
  target._o = markOnce;
  target._n = toNumber;
  target._s = toString;
  target._l = renderList;
  target._t = renderSlot;
  target._q = looseEqual;
  target._i = looseIndexOf;
  target._m = renderStatic;
  target._f = resolveFilter;
  target._k = checkKeyCodes;
  target._b = bindObjectProps;
  target._v = createTextVNode;
  target._e = createEmptyVNode;
  target._u = resolveScopedSlots;
  target._g = bindObjectListeners;
  target._d = bindDynamicKeys;
  target._p = prependModifier;
}

/*  */

function FunctionalRenderContext (
  data,
  props,
  children,
  parent,
  Ctor
) {
  var this$1 = this;

  var options = Ctor.options;
  // ensure the createElement function in functional components
  // gets a unique context - this is necessary for correct named slot check
  var contextVm;
  if (hasOwn(parent, '_uid')) {
    contextVm = Object.create(parent);
    // $flow-disable-line
    contextVm._original = parent;
  } else {
    // the context vm passed in is a functional context as well.
    // in this case we want to make sure we are able to get a hold to the
    // real context instance.
    contextVm = parent;
    // $flow-disable-line
    parent = parent._original;
  }
  var isCompiled = isTrue(options._compiled);
  var needNormalization = !isCompiled;

  this.data = data;
  this.props = props;
  this.children = children;
```

```
  this.parent = parent;
  this.listeners = data.on || emptyObject;
  this.injections = resolveInject(options.inject, parent);
  this.slots = function () {
    if (!this$1.$slots) {
      normalizeScopedSlots(
        data.scopedSlots,
        this$1.$slots = resolveSlots(children, parent)
      );
    }
    return this$1.$slots
  };

  Object.defineProperty(this, 'scopedSlots', ({
    enumerable: true,
    get: function get () {
      return normalizeScopedSlots(data.scopedSlots, this.slots())
    }
  }));

  // support for compiled functional template
  if (isCompiled) {
    // exposing $options for renderStatic()
    this.$options = options;
    // pre-resolve slots for renderSlot()
    this.$slots = this.slots();
    this.$scopedSlots = normalizeScopedSlots(data.scopedSlots, this.$slots);
  }

  if (options._scopeId) {
    this._c = function (a, b, c, d) {
      var vnode = createElement(contextVm, a, b, c, d, needNormalization);
      if (vnode && !Array.isArray(vnode)) {
        vnode.fnScopeId = options._scopeId;
        vnode.fnContext = parent;
      }
      return vnode
    };
  } else {
    this._c = function (a, b, c, d) { return createElement(contextVm, a, b, c, d, needNormali
  }
}

installRenderHelpers(FunctionalRenderContext.prototype);

function createFunctionalComponent (
  Ctor,
  propsData,
  data,
  contextVm,
  children
) {
  var options = Ctor.options;
  var props = {};
  var propOptions = options.props;
  if (isDef(propOptions)) {
    for (var key in propOptions) {
      props[key] = validateProp(key, propOptions, propsData || emptyObject);
    }
  } else {
    if (isDef(data.attrs)) { mergeProps(props, data.attrs); }
    if (isDef(data.props)) { mergeProps(props, data.props); }
```

```
  }

  var renderContext = new FunctionalRenderContext(
    data,
    props,
    children,
    contextVm,
    Ctor
  );

  var vnode = options.render.call(null, renderContext._c, renderContext);

  if (vnode instanceof VNode) {
    return cloneAndMarkFunctionalResult(vnode, data, renderContext.parent, options, renderCon
  } else if (Array.isArray(vnode)) {
    var vnodes = normalizeChildren(vnode) || [];
    var res = new Array(vnodes.length);
    for (var i = 0; i < vnodes.length; i++) {
      res[i] = cloneAndMarkFunctionalResult(vnodes[i], data, renderContext.parent, options, r
    }
    return res
  }
}

function cloneAndMarkFunctionalResult (vnode, data, contextVm, options, renderContext) {
  // #7817 clone node before setting fnContext, otherwise if the node is reused
  // (e.g. it was from a cached normal slot) the fnContext causes named slots
  // that should not be matched to match.
  var clone = cloneVNode(vnode);
  clone.fnContext = contextVm;
  clone.fnOptions = options;
  {
    (clone.devtoolsMeta = clone.devtoolsMeta || {}).renderContext = renderContext;
  }
  if (data.slot) {
    (clone.data || (clone.data = {})).slot = data.slot;
  }
  return clone
}

function mergeProps (to, from) {
  for (var key in from) {
    to[camelize(key)] = from[key];
  }
}

/*  */

/*  */

/*  */

/*  */

// inline hooks to be invoked on component VNodes during patch
var componentVNodeHooks = {
  init: function init (vnode, hydrating) {
    if (
      vnode.componentInstance &&
      !vnode.componentInstance._isDestroyed &&
      vnode.data.keepAlive
    ) {
```

```
      // kept-alive components, treat as a patch
      var mountedNode = vnode; // work around flow
      componentVNodeHooks.prepatch(mountedNode, mountedNode);
    } else {
      var child = vnode.componentInstance = createComponentInstanceForVnode(
        vnode,
        activeInstance
      );
      child.$mount(hydrating ? vnode.elm : undefined, hydrating);
    }
  },

  prepatch: function prepatch (oldVnode, vnode) {
    var options = vnode.componentOptions;
    var child = vnode.componentInstance = oldVnode.componentInstance;
    updateChildComponent(
      child,
      options.propsData, // updated props
      options.listeners, // updated listeners
      vnode, // new parent vnode
      options.children // new children
    );
  },

  insert: function insert (vnode) {
    var context = vnode.context;
    var componentInstance = vnode.componentInstance;
    if (!componentInstance._isMounted) {
      componentInstance._isMounted = true;
      callHook(componentInstance, 'mounted');
    }
    if (vnode.data.keepAlive) {
      if (context._isMounted) {
        // vue-router#1212
        // During updates, a kept-alive component's child components may
        // change, so directly walking the tree here may call activated hooks
        // on incorrect children. Instead we push them into a queue which will
        // be processed after the whole patch process ended.
        queueActivatedComponent(componentInstance);
      } else {
        activateChildComponent(componentInstance, true /* direct */);
      }
    }
  },

  destroy: function destroy (vnode) {
    var componentInstance = vnode.componentInstance;
    if (!componentInstance._isDestroyed) {
      if (!vnode.data.keepAlive) {
        componentInstance.$destroy();
      } else {
        deactivateChildComponent(componentInstance, true /* direct */);
      }
    }
  }
};

var hooksToMerge = Object.keys(componentVNodeHooks);

function createComponent (
  Ctor,
  data,
```

```
  context,
  children,
  tag
) {
  if (isUndef(Ctor)) {
    return
  }

  var baseCtor = context.$options._base;

  // plain options object: turn it into a constructor
  if (isObject(Ctor)) {
    Ctor = baseCtor.extend(Ctor);
  }

  // if at this stage it's not a constructor or an async component factory,
  // reject.
  if (typeof Ctor !== 'function') {
    {
      warn(("Invalid Component definition: " + (String(Ctor))), context);
    }
    return
  }

  // async component
  var asyncFactory;
  if (isUndef(Ctor.cid)) {
    asyncFactory = Ctor;
    Ctor = resolveAsyncComponent(asyncFactory, baseCtor);
    if (Ctor === undefined) {
      // return a placeholder node for async component, which is rendered
      // as a comment node but preserves all the raw information for the node.
      // the information will be used for async server-rendering and hydration.
      return createAsyncPlaceholder(
        asyncFactory,
        data,
        context,
        children,
        tag
      )
    }
  }

  data = data || {};

  // resolve constructor options in case global mixins are applied after
  // component constructor creation
  resolveConstructorOptions(Ctor);

  // transform component v-model data into props & events
  if (isDef(data.model)) {
    transformModel(Ctor.options, data);
  }

  // extract props
  var propsData = extractPropsFromVNodeData(data, Ctor, tag);

  // functional component
  if (isTrue(Ctor.options.functional)) {
    return createFunctionalComponent(Ctor, propsData, data, context, children)
  }
```

```
  // extract listeners, since these needs to be treated as
  // child component listeners instead of DOM listeners
  var listeners = data.on;
  // replace with listeners with .native modifier
  // so it gets processed during parent component patch.
  data.on = data.nativeOn;

  if (isTrue(Ctor.options.abstract)) {
    // abstract components do not keep anything
    // other than props & listeners & slot

    // work around flow
    var slot = data.slot;
    data = {};
    if (slot) {
      data.slot = slot;
    }
  }

  // install component management hooks onto the placeholder node
  installComponentHooks(data);

  // return a placeholder vnode
  var name = Ctor.options.name || tag;
  var vnode = new VNode(
    ("vue-component-" + (Ctor.cid) + (name ? ("-" + name) : '')),
    data, undefined, undefined, undefined, context,
    { Ctor: Ctor, propsData: propsData, listeners: listeners, tag: tag, children: children },
    asyncFactory
  );

  return vnode
}

function createComponentInstanceForVnode (
  vnode, // we know it's MountedComponentVNode but flow doesn't
  parent // activeInstance in lifecycle state
) {
  var options = {
    _isComponent: true,
    _parentVnode: vnode,
    parent: parent
  };
  // check inline-template render functions
  var inlineTemplate = vnode.data.inlineTemplate;
  if (isDef(inlineTemplate)) {
    options.render = inlineTemplate.render;
    options.staticRenderFns = inlineTemplate.staticRenderFns;
  }
  return new vnode.componentOptions.Ctor(options)
}

function installComponentHooks (data) {
  var hooks = data.hook || (data.hook = {});
  for (var i = 0; i < hooksToMerge.length; i++) {
    var key = hooksToMerge[i];
    var existing = hooks[key];
    var toMerge = componentVNodeHooks[key];
    if (existing !== toMerge && !(existing && existing._merged)) {
      hooks[key] = existing ? mergeHook$1(toMerge, existing) : toMerge;
    }
  }
```

```
}

function mergeHook$1 (f1, f2) {
  var merged = function (a, b) {
    // flow complains about extra args which is why we use any
    f1(a, b);
    f2(a, b);
  };
  merged._merged = true;
  return merged
}

// transform component v-model info (value and callback) into
// prop and event handler respectively.
function transformModel (options, data) {
  var prop = (options.model && options.model.prop) || 'value';
  var event = (options.model && options.model.event) || 'input'
  ;(data.attrs || (data.attrs = {}))[prop] = data.model.value;
  var on = data.on || (data.on = {});
  var existing = on[event];
  var callback = data.model.callback;
  if (isDef(existing)) {
    if (
      Array.isArray(existing)
        ? existing.indexOf(callback) === -1
        : existing !== callback
    ) {
      on[event] = [callback].concat(existing);
    }
  } else {
    on[event] = callback;
  }
}

/*  */

var SIMPLE_NORMALIZE = 1;
var ALWAYS_NORMALIZE = 2;

// wrapper function for providing a more flexible interface
// without getting yelled at by flow
function createElement (
  context,
  tag,
  data,
  children,
  normalizationType,
  alwaysNormalize
) {
  if (Array.isArray(data) || isPrimitive(data)) {
    normalizationType = children;
    children = data;
    data = undefined;
  }
  if (isTrue(alwaysNormalize)) {
    normalizationType = ALWAYS_NORMALIZE;
  }
  return _createElement(context, tag, data, children, normalizationType)
}

function _createElement (
  context,
```

```
  tag,
  data,
  children,
  normalizationType
) {
  if (isDef(data) && isDef((data).__ob__)) {
    warn(
      "Avoid using observed data object as vnode data: " + (JSON.stringify(data)) + "\n" +
      'Always create fresh vnode data objects in each render!',
      context
    );
    return createEmptyVNode()
  }
  // object syntax in v-bind
  if (isDef(data) && isDef(data.is)) {
    tag = data.is;
  }
  if (!tag) {
    // in case of component :is set to falsy value
    return createEmptyVNode()
  }
  // warn against non-primitive key
  if (isDef(data) && isDef(data.key) && !isPrimitive(data.key)
  ) {
    {
      warn(
        'Avoid using non-primitive value as key, ' +
        'use string/number value instead.',
        context
      );
    }
  }
  // support single function children as default scoped slot
  if (Array.isArray(children) &&
    typeof children[0] === 'function'
  ) {
    data = data || {};
    data.scopedSlots = { default: children[0] };
    children.length = 0;
  }
  if (normalizationType === ALWAYS_NORMALIZE) {
    children = normalizeChildren(children);
  } else if (normalizationType === SIMPLE_NORMALIZE) {
    children = simpleNormalizeChildren(children);
  }
  var vnode, ns;
  if (typeof tag === 'string') {
    var Ctor;
    ns = (context.$vnode && context.$vnode.ns) || config.getTagNamespace(tag);
    if (config.isReservedTag(tag)) {
      // platform built-in elements
      vnode = new VNode(
        config.parsePlatformTagName(tag), data, children,
        undefined, undefined, context
      );
    } else if ((!data || !data.pre) && isDef(Ctor = resolveAsset(context.$options, 'component
      // component
      vnode = createComponent(Ctor, data, context, children, tag);
    } else {
      // unknown or unlisted namespaced elements
      // check at runtime because it may get assigned a namespace when its
      // parent normalizes children
```

```
      vnode = new VNode(
        tag, data, children,
        undefined, undefined, context
      );
    }
  } else {
    // direct component options / constructor
    vnode = createComponent(tag, data, context, children);
  }
  if (Array.isArray(vnode)) {
    return vnode
  } else if (isDef(vnode)) {
    if (isDef(ns)) { applyNS(vnode, ns); }
    if (isDef(data)) { registerDeepBindings(data); }
    return vnode
  } else {
    return createEmptyVNode()
  }
}

function applyNS (vnode, ns, force) {
  vnode.ns = ns;
  if (vnode.tag === 'foreignObject') {
    // use default namespace inside foreignObject
    ns = undefined;
    force = true;
  }
  if (isDef(vnode.children)) {
    for (var i = 0, l = vnode.children.length; i < l; i++) {
      var child = vnode.children[i];
      if (isDef(child.tag) && (
        isUndef(child.ns) || (isTrue(force) && child.tag !== 'svg'))) {
        applyNS(child, ns, force);
      }
    }
  }
}

// ref #5318
// necessary to ensure parent re-render when deep bindings like :style and
// :class are used on slot nodes
function registerDeepBindings (data) {
  if (isObject(data.style)) {
    traverse(data.style);
  }
  if (isObject(data.class)) {
    traverse(data.class);
  }
}

/*  */

function initRender (vm) {
  vm._vnode = null; // the root of the child tree
  vm._staticTrees = null; // v-once cached trees
  var options = vm.$options;
  var parentVnode = vm.$vnode = options._parentVnode; // the placeholder node in parent tree
  var renderContext = parentVnode && parentVnode.context;
  vm.$slots = resolveSlots(options._renderChildren, renderContext);
  vm.$scopedSlots = emptyObject;
  // bind the createElement fn to this instance
  // so that we get proper render context inside it.
```

```
  // args order: tag, data, children, normalizationType, alwaysNormalize
  // internal version is used by render functions compiled from templates
  vm._c = function (a, b, c, d) { return createElement(vm, a, b, c, d, false); };
  // normalization is always applied for the public version, used in
  // user-written render functions.
  vm.$createElement = function (a, b, c, d) { return createElement(vm, a, b, c, d, true); };

  // $attrs & $listeners are exposed for easier HOC creation.
  // they need to be reactive so that HOCs using them are always updated
  var parentData = parentVnode && parentVnode.data;

  /* istanbul ignore else */
  {
    defineReactive$$1(vm, '$attrs', parentData && parentData.attrs || emptyObject, function (
      !isUpdatingChildComponent && warn("$attrs is readonly.", vm);
    }, true);
    defineReactive$$1(vm, '$listeners', options._parentListeners || emptyObject, function ()
      !isUpdatingChildComponent && warn("$listeners is readonly.", vm);
    }, true);
  }
}

var currentRenderingInstance = null;

function renderMixin (Vue) {
  // install runtime convenience helpers
  installRenderHelpers(Vue.prototype);

  Vue.prototype.$nextTick = function (fn) {
    return nextTick(fn, this)
  };

  Vue.prototype._render = function () {
    var vm = this;
    var ref = vm.$options;
    var render = ref.render;
    var _parentVnode = ref._parentVnode;

    if (_parentVnode) {
      vm.$scopedSlots = normalizeScopedSlots(
        _parentVnode.data.scopedSlots,
        vm.$slots,
        vm.$scopedSlots
      );
    }

    // set parent vnode. this allows render functions to have access
    // to the data on the placeholder node.
    vm.$vnode = _parentVnode;
    // render self
    var vnode;
    try {
      // There's no need to maintain a stack becaues all render fns are called
      // separately from one another. Nested component's render fns are called
      // when parent component is patched.
      currentRenderingInstance = vm;
      vnode = render.call(vm._renderProxy, vm.$createElement);
    } catch (e) {
      handleError(e, vm, "render");
      // return error render result,
      // or previous vnode to prevent render error causing blank component
      /* istanbul ignore else */
```

```
      if (vm.$options.renderError) {
        try {
          vnode = vm.$options.renderError.call(vm._renderProxy, vm.$createElement, e);
        } catch (e) {
          handleError(e, vm, "renderError");
          vnode = vm._vnode;
        }
      } else {
        vnode = vm._vnode;
      }
    } finally {
      currentRenderingInstance = null;
    }
    // if the returned array contains only a single node, allow it
    if (Array.isArray(vnode) && vnode.length === 1) {
      vnode = vnode[0];
    }
    // return empty vnode in case the render function errored out
    if (!(vnode instanceof VNode)) {
      if (Array.isArray(vnode)) {
        warn(
          'Multiple root nodes returned from render function. Render function ' +
          'should return a single root node.',
          vm
        );
      }
      vnode = createEmptyVNode();
    }
    // set parent
    vnode.parent = _parentVnode;
    return vnode
  };
}

/*  */

function ensureCtor (comp, base) {
  if (
    comp.__esModule ||
    (hasSymbol && comp[Symbol.toStringTag] === 'Module')
  ) {
    comp = comp.default;
  }
  return isObject(comp)
    ? base.extend(comp)
    : comp
}

function createAsyncPlaceholder (
  factory,
  data,
  context,
  children,
  tag
) {
  var node = createEmptyVNode();
  node.asyncFactory = factory;
  node.asyncMeta = { data: data, context: context, children: children, tag: tag };
  return node
}

function resolveAsyncComponent (
```

```
  factory,
  baseCtor
) {
  if (isTrue(factory.error) && isDef(factory.errorComp)) {
    return factory.errorComp
  }

  if (isDef(factory.resolved)) {
    return factory.resolved
  }

  var owner = currentRenderingInstance;
  if (owner && isDef(factory.owners) && factory.owners.indexOf(owner) === -1) {
    // already pending
    factory.owners.push(owner);
  }

  if (isTrue(factory.loading) && isDef(factory.loadingComp)) {
    return factory.loadingComp
  }

  if (owner && !isDef(factory.owners)) {
    var owners = factory.owners = [owner];
    var sync = true;
    var timerLoading = null;
    var timerTimeout = null

    ;(owner).$on('hook:destroyed', function () { return remove(owners, owner); });

    var forceRender = function (renderCompleted) {
      for (var i = 0, l = owners.length; i < l; i++) {
        (owners[i]).$forceUpdate();
      }

      if (renderCompleted) {
        owners.length = 0;
        if (timerLoading !== null) {
          clearTimeout(timerLoading);
          timerLoading = null;
        }
        if (timerTimeout !== null) {
          clearTimeout(timerTimeout);
          timerTimeout = null;
        }
      }
    };

    var resolve = once(function (res) {
      // cache resolved
      factory.resolved = ensureCtor(res, baseCtor);
      // invoke callbacks only if this is not a synchronous resolve
      // (async resolves are shimmed as synchronous during SSR)
      if (!sync) {
        forceRender(true);
      } else {
        owners.length = 0;
      }
    });

    var reject = once(function (reason) {
      warn(
        "Failed to resolve async component: " + (String(factory)) +
```

```
      (reason ? ("\nReason: " + reason) : '')
    );
    if (isDef(factory.errorComp)) {
      factory.error = true;
      forceRender(true);
    }
  });

  var res = factory(resolve, reject);

  if (isObject(res)) {
    if (isPromise(res)) {
      // () => Promise
      if (isUndef(factory.resolved)) {
        res.then(resolve, reject);
      }
    } else if (isPromise(res.component)) {
      res.component.then(resolve, reject);

      if (isDef(res.error)) {
        factory.errorComp = ensureCtor(res.error, baseCtor);
      }

      if (isDef(res.loading)) {
        factory.loadingComp = ensureCtor(res.loading, baseCtor);
        if (res.delay === 0) {
          factory.loading = true;
        } else {
          timerLoading = setTimeout(function () {
            timerLoading = null;
            if (isUndef(factory.resolved) && isUndef(factory.error)) {
              factory.loading = true;
              forceRender(false);
            }
          }, res.delay || 200);
        }
      }

      if (isDef(res.timeout)) {
        timerTimeout = setTimeout(function () {
          timerTimeout = null;
          if (isUndef(factory.resolved)) {
            reject(
              "timeout (" + (res.timeout) + "ms)"
            );
          }
        }, res.timeout);
      }
    }
  }

  sync = false;
  // return in case resolved synchronously
  return factory.loading
    ? factory.loadingComp
    : factory.resolved
 }
}

/*  */

function isAsyncPlaceholder (node) {
```

```
  return node.isComment && node.asyncFactory
}

/*  */

function getFirstComponentChild (children) {
  if (Array.isArray(children)) {
    for (var i = 0; i < children.length; i++) {
      var c = children[i];
      if (isDef(c) && (isDef(c.componentOptions) || isAsyncPlaceholder(c))) {
        return c
      }
    }
  }
}

/*  */

/*  */

function initEvents (vm) {
  vm._events = Object.create(null);
  vm._hasHookEvent = false;
  // init parent attached events
  var listeners = vm.$options._parentListeners;
  if (listeners) {
    updateComponentListeners(vm, listeners);
  }
}

var target;

function add (event, fn) {
  target.$on(event, fn);
}

function remove$1 (event, fn) {
  target.$off(event, fn);
}

function createOnceHandler (event, fn) {
  var _target = target;
  return function onceHandler () {
    var res = fn.apply(null, arguments);
    if (res !== null) {
      _target.$off(event, onceHandler);
    }
  }
}

function updateComponentListeners (
  vm,
  listeners,
  oldListeners
) {
  target = vm;
  updateListeners(listeners, oldListeners || {}, add, remove$1, createOnceHandler, vm);
  target = undefined;
}

function eventsMixin (Vue) {
  var hookRE = /^hook:/;
```

```
Vue.prototype.$on = function (event, fn) {
  var vm = this;
  if (Array.isArray(event)) {
    for (var i = 0, l = event.length; i < l; i++) {
      vm.$on(event[i], fn);
    }
  } else {
    (vm._events[event] || (vm._events[event] = [])).push(fn);
    // optimize hook:event cost by using a boolean flag marked at registration
    // instead of a hash lookup
    if (hookRE.test(event)) {
      vm._hasHookEvent = true;
    }
  }
  return vm
};

Vue.prototype.$once = function (event, fn) {
  var vm = this;
  function on () {
    vm.$off(event, on);
    fn.apply(vm, arguments);
  }
  on.fn = fn;
  vm.$on(event, on);
  return vm
};

Vue.prototype.$off = function (event, fn) {
  var vm = this;
  // all
  if (!arguments.length) {
    vm._events = Object.create(null);
    return vm
  }
  // array of events
  if (Array.isArray(event)) {
    for (var i$1 = 0, l = event.length; i$1 < l; i$1++) {
      vm.$off(event[i$1], fn);
    }
    return vm
  }
  // specific event
  var cbs = vm._events[event];
  if (!cbs) {
    return vm
  }
  if (!fn) {
    vm._events[event] = null;
    return vm
  }
  // specific handler
  var cb;
  var i = cbs.length;
  while (i--) {
    cb = cbs[i];
    if (cb === fn || cb.fn === fn) {
      cbs.splice(i, 1);
      break
    }
  }
  return vm
```

```
  };

  Vue.prototype.$emit = function (event) {
    var vm = this;
    {
      var lowerCaseEvent = event.toLowerCase();
      if (lowerCaseEvent !== event && vm._events[lowerCaseEvent]) {
        tip(
          "Event \"" + lowerCaseEvent + "\" is emitted in component " +
          (formatComponentName(vm)) + " but the handler is registered for \"" + event + "\".
          "Note that HTML attributes are case-insensitive and you cannot use " +
          "v-on to listen to camelCase events when using in-DOM templates. " +
          "You should probably use \"" + (hyphenate(event)) + "\" instead of \"" + event + "\
        );
      }
    }
    var cbs = vm._events[event];
    if (cbs) {
      cbs = cbs.length > 1 ? toArray(cbs) : cbs;
      var args = toArray(arguments, 1);
      var info = "event handler for \"" + event + "\"";
      for (var i = 0, l = cbs.length; i < l; i++) {
        invokeWithErrorHandling(cbs[i], vm, args, vm, info);
      }
    }
    return vm
  };
}

/*  */

var activeInstance = null;
var isUpdatingChildComponent = false;

function setActiveInstance(vm) {
  var prevActiveInstance = activeInstance;
  activeInstance = vm;
  return function () {
    activeInstance = prevActiveInstance;
  }
}

function initLifecycle (vm) {
  var options = vm.$options;

  // locate first non-abstract parent
  var parent = options.parent;
  if (parent && !options.abstract) {
    while (parent.$options.abstract && parent.$parent) {
      parent = parent.$parent;
    }
    parent.$children.push(vm);
  }

  vm.$parent = parent;
  vm.$root = parent ? parent.$root : vm;

  vm.$children = [];
  vm.$refs = {};

  vm._watcher = null;
  vm._inactive = null;
```

```
  vm._directInactive = false;
  vm._isMounted = false;
  vm._isDestroyed = false;
  vm._isBeingDestroyed = false;
}

function lifecycleMixin (Vue) {
  Vue.prototype._update = function (vnode, hydrating) {
    var vm = this;
    var prevEl = vm.$el;
    var prevVnode = vm._vnode;
    var restoreActiveInstance = setActiveInstance(vm);
    vm._vnode = vnode;
    // Vue.prototype.__patch__ is injected in entry points
    // based on the rendering backend used.
    if (!prevVnode) {
      // initial render
      vm.$el = vm.__patch__(vm.$el, vnode, hydrating, false /* removeOnly */);
    } else {
      // updates
      vm.$el = vm.__patch__(prevVnode, vnode);
    }
    restoreActiveInstance();
    // update __vue__ reference
    if (prevEl) {
      prevEl.__vue__ = null;
    }
    if (vm.$el) {
      vm.$el.__vue__ = vm;
    }
    // if parent is an HOC, update its $el as well
    if (vm.$vnode && vm.$parent && vm.$vnode === vm.$parent._vnode) {
      vm.$parent.$el = vm.$el;
    }
    // updated hook is called by the scheduler to ensure that children are
    // updated in a parent's updated hook.
  };

  Vue.prototype.$forceUpdate = function () {
    var vm = this;
    if (vm._watcher) {
      vm._watcher.update();
    }
  };

  Vue.prototype.$destroy = function () {
    var vm = this;
    if (vm._isBeingDestroyed) {
      return
    }
    callHook(vm, 'beforeDestroy');
    vm._isBeingDestroyed = true;
    // remove self from parent
    var parent = vm.$parent;
    if (parent && !parent._isBeingDestroyed && !vm.$options.abstract) {
      remove(parent.$children, vm);
    }
    // teardown watchers
    if (vm._watcher) {
      vm._watcher.teardown();
    }
    var i = vm._watchers.length;
```

```
    while (i--) {
      vm._watchers[i].teardown();
    }
    // remove reference from data ob
    // frozen object may not have observer.
    if (vm._data.__ob__) {
      vm._data.__ob__.vmCount--;
    }
    // call the last hook...
    vm._isDestroyed = true;
    // invoke destroy hooks on current rendered tree
    vm.__patch__(vm._vnode, null);
    // fire destroyed hook
    callHook(vm, 'destroyed');
    // turn off all instance listeners.
    vm.$off();
    // remove __vue__ reference
    if (vm.$el) {
      vm.$el.__vue__ = null;
    }
    // release circular reference (#6759)
    if (vm.$vnode) {
      vm.$vnode.parent = null;
    }
  };
}

function mountComponent (
  vm,
  el,
  hydrating
) {
  vm.$el = el;
  if (!vm.$options.render) {
    vm.$options.render = createEmptyVNode;
    {
      /* istanbul ignore if */
      if ((vm.$options.template && vm.$options.template.charAt(0) !== '#') ||
        vm.$options.el || el) {
        warn(
          'You are using the runtime-only build of Vue where the template ' +
          'compiler is not available. Either pre-compile the templates into ' +
          'render functions, or use the compiler-included build.',
          vm
        );
      } else {
        warn(
          'Failed to mount component: template or render function not defined.',
          vm
        );
      }
    }
  }
  callHook(vm, 'beforeMount');

  var updateComponent;
  /* istanbul ignore if */
  if (config.performance && mark) {
    updateComponent = function () {
      var name = vm._name;
      var id = vm._uid;
      var startTag = "vue-perf-start:" + id;
```

```
      var endTag = "vue-perf-end:" + id;

      mark(startTag);
      var vnode = vm._render();
      mark(endTag);
      measure(("vue " + name + " render"), startTag, endTag);

      mark(startTag);
      vm._update(vnode, hydrating);
      mark(endTag);
      measure(("vue " + name + " patch"), startTag, endTag);
    };
  } else {
    updateComponent = function () {
      vm._update(vm._render(), hydrating);
    };
  }

  // we set this to vm._watcher inside the watcher's constructor
  // since the watcher's initial patch may call $forceUpdate (e.g. inside child
  // component's mounted hook), which relies on vm._watcher being already defined
  new Watcher(vm, updateComponent, noop, {
    before: function before () {
      if (vm._isMounted && !vm._isDestroyed) {
        callHook(vm, 'beforeUpdate');
      }
    }
  }, true /* isRenderWatcher */);
  hydrating = false;

  // manually mounted instance, call mounted on self
  // mounted is called for render-created child components in its inserted hook
  if (vm.$vnode == null) {
    vm._isMounted = true;
    callHook(vm, 'mounted');
  }
  return vm
}

function updateChildComponent (
  vm,
  propsData,
  listeners,
  parentVnode,
  renderChildren
) {
  {
    isUpdatingChildComponent = true;
  }

  // determine whether component has slot children
  // we need to do this before overwriting $options._renderChildren.

  // check if there are dynamic scopedSlots (hand-written or compiled but with
  // dynamic slot names). Static scoped slots compiled from template has the
  // "$stable" marker.
  var newScopedSlots = parentVnode.data.scopedSlots;
  var oldScopedSlots = vm.$scopedSlots;
  var hasDynamicScopedSlot = !!(
    (newScopedSlots && !newScopedSlots.$stable) ||
    (oldScopedSlots !== emptyObject && !oldScopedSlots.$stable) ||
    (newScopedSlots && vm.$scopedSlots.$key !== newScopedSlots.$key)
```

```
  );

  // Any static slot children from the parent may have changed during parent's
  // update. Dynamic scoped slots may also have changed. In such cases, a forced
  // update is necessary to ensure correctness.
  var needsForceUpdate = !!(
    renderChildren ||               // has new static slots
    vm.$options._renderChildren ||  // has old static slots
    hasDynamicScopedSlot
  );

  vm.$options._parentVnode = parentVnode;
  vm.$vnode = parentVnode; // update vm's placeholder node without re-render

  if (vm._vnode) { // update child tree's parent
    vm._vnode.parent = parentVnode;
  }
  vm.$options._renderChildren = renderChildren;

  // update $attrs and $listeners hash
  // these are also reactive so they may trigger child update if the child
  // used them during render
  vm.$attrs = parentVnode.data.attrs || emptyObject;
  vm.$listeners = listeners || emptyObject;

  // update props
  if (propsData && vm.$options.props) {
    toggleObserving(false);
    var props = vm._props;
    var propKeys = vm.$options._propKeys || [];
    for (var i = 0; i < propKeys.length; i++) {
      var key = propKeys[i];
      var propOptions = vm.$options.props; // wtf flow?
      props[key] = validateProp(key, propOptions, propsData, vm);
    }
    toggleObserving(true);
    // keep a copy of raw propsData
    vm.$options.propsData = propsData;
  }

  // update listeners
  listeners = listeners || emptyObject;
  var oldListeners = vm.$options._parentListeners;
  vm.$options._parentListeners = listeners;
  updateComponentListeners(vm, listeners, oldListeners);

  // resolve slots + force update if has children
  if (needsForceUpdate) {
    vm.$slots = resolveSlots(renderChildren, parentVnode.context);
    vm.$forceUpdate();
  }

  {
    isUpdatingChildComponent = false;
  }
}

function isInInactiveTree (vm) {
  while (vm && (vm = vm.$parent)) {
    if (vm._inactive) { return true }
  }
  return false
```

```
}

function activateChildComponent (vm, direct) {
  if (direct) {
    vm._directInactive = false;
    if (isInInactiveTree(vm)) {
      return
    }
  } else if (vm._directInactive) {
    return
  }
  if (vm._inactive || vm._inactive === null) {
    vm._inactive = false;
    for (var i = 0; i < vm.$children.length; i++) {
      activateChildComponent(vm.$children[i]);
    }
    callHook(vm, 'activated');
  }
}

function deactivateChildComponent (vm, direct) {
  if (direct) {
    vm._directInactive = true;
    if (isInInactiveTree(vm)) {
      return
    }
  }
  if (!vm._inactive) {
    vm._inactive = true;
    for (var i = 0; i < vm.$children.length; i++) {
      deactivateChildComponent(vm.$children[i]);
    }
    callHook(vm, 'deactivated');
  }
}

function callHook (vm, hook) {
  // #7573 disable dep collection when invoking lifecycle hooks
  pushTarget();
  var handlers = vm.$options[hook];
  var info = hook + " hook";
  if (handlers) {
    for (var i = 0, j = handlers.length; i < j; i++) {
      invokeWithErrorHandling(handlers[i], vm, null, vm, info);
    }
  }
  if (vm._hasHookEvent) {
    vm.$emit('hook:' + hook);
  }
  popTarget();
}

/*  */

var MAX_UPDATE_COUNT = 100;

var queue = [];
var activatedChildren = [];
var has = {};
var circular = {};
var waiting = false;
var flushing = false;
```

```
var index = 0;

/**
 * Reset the scheduler's state.
 */
function resetSchedulerState () {
  index = queue.length = activatedChildren.length = 0;
  has = {};
  {
    circular = {};
  }
  waiting = flushing = false;
}

// Async edge case #6566 requires saving the timestamp when event listeners are
// attached. However, calling performance.now() has a perf overhead especially
// if the page has thousands of event listeners. Instead, we take a timestamp
// every time the scheduler flushes and use that for all event listeners
// attached during that flush.
var currentFlushTimestamp = 0;

// Async edge case fix requires storing an event listener's attach timestamp.
var getNow = Date.now;

// Determine what event timestamp the browser is using. Annoyingly, the
// timestamp can either be hi-res (relative to page load) or low-res
// (relative to UNIX epoch), so in order to compare time we have to use the
// same timestamp type when saving the flush timestamp.
// All IE versions use low-res event timestamps, and have problematic clock
// implementations (#9632)
if (inBrowser && !isIE) {
  var performance = window.performance;
  if (
    performance &&
    typeof performance.now === 'function' &&
    getNow() > document.createEvent('Event').timeStamp
  ) {
    // if the event timestamp, although evaluated AFTER the Date.now(), is
    // smaller than it, it means the event is using a hi-res timestamp,
    // and we need to use the hi-res version for event listener timestamps as
    // well.
    getNow = function () { return performance.now(); };
  }
}

/**
 * Flush both queues and run the watchers.
 */
function flushSchedulerQueue () {
  currentFlushTimestamp = getNow();
  flushing = true;
  var watcher, id;

  // Sort queue before flush.
  // This ensures that:
  // 1. Components are updated from parent to child. (because parent is always
  //    created before the child)
  // 2. A component's user watchers are run before its render watcher (because
  //    user watchers are created before the render watcher)
  // 3. If a component is destroyed during a parent component's watcher run,
  //    its watchers can be skipped.
  queue.sort(function (a, b) { return a.id - b.id; });
```

```
  // do not cache length because more watchers might be pushed
  // as we run existing watchers
  for (index = 0; index < queue.length; index++) {
    watcher = queue[index];
    if (watcher.before) {
      watcher.before();
    }
    id = watcher.id;
    has[id] = null;
    watcher.run();
    // in dev build, check and stop circular updates.
    if (has[id] != null) {
      circular[id] = (circular[id] || 0) + 1;
      if (circular[id] > MAX_UPDATE_COUNT) {
        warn(
          'You may have an infinite update loop ' + (
            watcher.user
              ? ("in watcher with expression \"" + (watcher.expression) + "\"")
              : "in a component render function."
          ),
          watcher.vm
        );
        break
      }
    }
  }

  // keep copies of post queues before resetting state
  var activatedQueue = activatedChildren.slice();
  var updatedQueue = queue.slice();

  resetSchedulerState();

  // call component updated and activated hooks
  callActivatedHooks(activatedQueue);
  callUpdatedHooks(updatedQueue);

  // devtool hook
  /* istanbul ignore if */
  if (devtools && config.devtools) {
    devtools.emit('flush');
  }
}

function callUpdatedHooks (queue) {
  var i = queue.length;
  while (i--) {
    var watcher = queue[i];
    var vm = watcher.vm;
    if (vm._watcher === watcher && vm._isMounted && !vm._isDestroyed) {
      callHook(vm, 'updated');
    }
  }
}

/**
 * Queue a kept-alive component that was activated during patch.
 * The queue will be processed after the entire tree has been patched.
 */
function queueActivatedComponent (vm) {
  // setting _inactive to false here so that a render function can
```

```
  // rely on checking whether it's in an inactive tree (e.g. router-view)
  vm._inactive = false;
  activatedChildren.push(vm);
}

function callActivatedHooks (queue) {
  for (var i = 0; i < queue.length; i++) {
    queue[i]._inactive = true;
    activateChildComponent(queue[i], true /* true */);
  }
}

/**
 * Push a watcher into the watcher queue.
 * Jobs with duplicate IDs will be skipped unless it's
 * pushed when the queue is being flushed.
 */
function queueWatcher (watcher) {
  var id = watcher.id;
  if (has[id] == null) {
    has[id] = true;
    if (!flushing) {
      queue.push(watcher);
    } else {
      // if already flushing, splice the watcher based on its id
      // if already past its id, it will be run next immediately.
      var i = queue.length - 1;
      while (i > index && queue[i].id > watcher.id) {
        i--;
      }
      queue.splice(i + 1, 0, watcher);
    }
    // queue the flush
    if (!waiting) {
      waiting = true;

      if (!config.async) {
        flushSchedulerQueue();
        return
      }
      nextTick(flushSchedulerQueue);
    }
  }
}

/*  */



var uid$2 = 0;

/**
 * A watcher parses an expression, collects dependencies,
 * and fires callback when the expression value changes.
 * This is used for both the $watch() api and directives.
 */
var Watcher = function Watcher (
  vm,
  expOrFn,
  cb,
  options,
  isRenderWatcher
```

```
) {
  this.vm = vm;
  if (isRenderWatcher) {
    vm._watcher = this;
  }
  vm._watchers.push(this);
  // options
  if (options) {
    this.deep = !!options.deep;
    this.user = !!options.user;
    this.lazy = !!options.lazy;
    this.sync = !!options.sync;
    this.before = options.before;
  } else {
    this.deep = this.user = this.lazy = this.sync = false;
  }
  this.cb = cb;
  this.id = ++uid$2; // uid for batching
  this.active = true;
  this.dirty = this.lazy; // for lazy watchers
  this.deps = [];
  this.newDeps = [];
  this.depIds = new _Set();
  this.newDepIds = new _Set();
  this.expression = expOrFn.toString();
  // parse expression for getter
  if (typeof expOrFn === 'function') {
    this.getter = expOrFn;
  } else {
    this.getter = parsePath(expOrFn);
    if (!this.getter) {
      this.getter = noop;
      warn(
        "Failed watching path: \"" + expOrFn + "\" " +
        'Watcher only accepts simple dot-delimited paths. ' +
        'For full control, use a function instead.',
        vm
      );
    }
  }
  this.value = this.lazy
    ? undefined
    : this.get();
};

/**
 * Evaluate the getter, and re-collect dependencies.
 */
Watcher.prototype.get = function get () {
  pushTarget(this);
  var value;
  var vm = this.vm;
  try {
    value = this.getter.call(vm, vm);
  } catch (e) {
    if (this.user) {
      handleError(e, vm, ("getter for watcher \"" + (this.expression) + "\""));
    } else {
      throw e
    }
  } finally {
    // "touch" every property so they are all tracked as
```

```
    // dependencies for deep watching
    if (this.deep) {
      traverse(value);
    }
    popTarget();
    this.cleanupDeps();
  }
  return value
};

/**
 * Add a dependency to this directive.
 */
Watcher.prototype.addDep = function addDep (dep) {
  var id = dep.id;
  if (!this.newDepIds.has(id)) {
    this.newDepIds.add(id);
    this.newDeps.push(dep);
    if (!this.depIds.has(id)) {
      dep.addSub(this);
    }
  }
};

/**
 * Clean up for dependency collection.
 */
Watcher.prototype.cleanupDeps = function cleanupDeps () {
  var i = this.deps.length;
  while (i--) {
    var dep = this.deps[i];
    if (!this.newDepIds.has(dep.id)) {
      dep.removeSub(this);
    }
  }
  var tmp = this.depIds;
  this.depIds = this.newDepIds;
  this.newDepIds = tmp;
  this.newDepIds.clear();
  tmp = this.deps;
  this.deps = this.newDeps;
  this.newDeps = tmp;
  this.newDeps.length = 0;
};

/**
 * Subscriber interface.
 * Will be called when a dependency changes.
 */
Watcher.prototype.update = function update () {
  /* istanbul ignore else */
  if (this.lazy) {
    this.dirty = true;
  } else if (this.sync) {
    this.run();
  } else {
    queueWatcher(this);
  }
};

/**
 * Scheduler job interface.
```

```
 * Will be called by the scheduler.
 */
Watcher.prototype.run = function run () {
  if (this.active) {
    var value = this.get();
    if (
      value !== this.value ||
      // Deep watchers and watchers on Object/Arrays should fire even
      // when the value is the same, because the value may
      // have mutated.
      isObject(value) ||
      this.deep
    ) {
      // set new value
      var oldValue = this.value;
      this.value = value;
      if (this.user) {
        try {
          this.cb.call(this.vm, value, oldValue);
        } catch (e) {
          handleError(e, this.vm, ("callback for watcher \"" + (this.expression) + "\""));
        }
      } else {
        this.cb.call(this.vm, value, oldValue);
      }
    }
  }
};

/**
 * Evaluate the value of the watcher.
 * This only gets called for lazy watchers.
 */
Watcher.prototype.evaluate = function evaluate () {
  this.value = this.get();
  this.dirty = false;
};

/**
 * Depend on all deps collected by this watcher.
 */
Watcher.prototype.depend = function depend () {
  var i = this.deps.length;
  while (i--) {
    this.deps[i].depend();
  }
};

/**
 * Remove self from all dependencies' subscriber list.
 */
Watcher.prototype.teardown = function teardown () {
  if (this.active) {
    // remove self from vm's watcher list
    // this is a somewhat expensive operation so we skip it
    // if the vm is being destroyed.
    if (!this.vm._isBeingDestroyed) {
      remove(this.vm._watchers, this);
    }
    var i = this.deps.length;
    while (i--) {
      this.deps[i].removeSub(this);
```

```
    }
    this.active = false;
  }
};

/*  */

var sharedPropertyDefinition = {
  enumerable: true,
  configurable: true,
  get: noop,
  set: noop
};

function proxy (target, sourceKey, key) {
  sharedPropertyDefinition.get = function proxyGetter () {
    return this[sourceKey][key]
  };
  sharedPropertyDefinition.set = function proxySetter (val) {
    this[sourceKey][key] = val;
  };
  Object.defineProperty(target, key, sharedPropertyDefinition);
}

function initState (vm) {
  vm._watchers = [];
  var opts = vm.$options;
  if (opts.props) { initProps(vm, opts.props); }
  if (opts.methods) { initMethods(vm, opts.methods); }
  if (opts.data) {
    initData(vm);
  } else {
    observe(vm._data = {}, true /* asRootData */);
  }
  if (opts.computed) { initComputed(vm, opts.computed); }
  if (opts.watch && opts.watch !== nativeWatch) {
    initWatch(vm, opts.watch);
  }
}

function initProps (vm, propsOptions) {
  var propsData = vm.$options.propsData || {};
  var props = vm._props = {};
  // cache prop keys so that future props updates can iterate using Array
  // instead of dynamic object key enumeration.
  var keys = vm.$options._propKeys = [];
  var isRoot = !vm.$parent;
  // root instance props should be converted
  if (!isRoot) {
    toggleObserving(false);
  }
  var loop = function ( key ) {
    keys.push(key);
    var value = validateProp(key, propsOptions, propsData, vm);
    /* istanbul ignore else */
    {
      var hyphenatedKey = hyphenate(key);
      if (isReservedAttribute(hyphenatedKey) ||
          config.isReservedAttr(hyphenatedKey)) {
        warn(
          ("\"" + hyphenatedKey + "\" is a reserved attribute and cannot be used as component
          vm
```

```
        );
      }
      defineReactive$$1(props, key, value, function () {
        if (!isRoot && !isUpdatingChildComponent) {
          warn(
            "Avoid mutating a prop directly since the value will be " +
            "overwritten whenever the parent component re-renders. " +
            "Instead, use a data or computed property based on the prop's " +
            "value. Prop being mutated: \"" + key + "\"",
            vm
          );
        }
      });
    }
    // static props are already proxied on the component's prototype
    // during Vue.extend(). We only need to proxy props defined at
    // instantiation here.
    if (!(key in vm)) {
      proxy(vm, "_props", key);
    }
  };

  for (var key in propsOptions) loop( key );
  toggleObserving(true);
}

function initData (vm) {
  var data = vm.$options.data;
  data = vm._data = typeof data === 'function'
    ? getData(data, vm)
    : data || {};
  if (!isPlainObject(data)) {
    data = {};
    warn(
      'data functions should return an object:\n' +
      'https://vuejs.org/v2/guide/components.html#data-Must-Be-a-Function',
      vm
    );
  }
  // proxy data on instance
  var keys = Object.keys(data);
  var props = vm.$options.props;
  var methods = vm.$options.methods;
  var i = keys.length;
  while (i--) {
    var key = keys[i];
    {
      if (methods && hasOwn(methods, key)) {
        warn(
          ("Method \"" + key + "\" has already been defined as a data property."),
          vm
        );
      }
    }
    if (props && hasOwn(props, key)) {
      warn(
        "The data property \"" + key + "\" is already declared as a prop. " +
        "Use prop default value instead.",
        vm
      );
    } else if (!isReserved(key)) {
      proxy(vm, "_data", key);
```

```
    }
  }
  // observe data
  observe(data, true /* asRootData */);
}

function getData (data, vm) {
  // #7573 disable dep collection when invoking data getters
  pushTarget();
  try {
    return data.call(vm, vm)
  } catch (e) {
    handleError(e, vm, "data()");
    return {}
  } finally {
    popTarget();
  }
}

var computedWatcherOptions = { lazy: true };

function initComputed (vm, computed) {
  // $flow-disable-line
  var watchers = vm._computedWatchers = Object.create(null);
  // computed properties are just getters during SSR
  var isSSR = isServerRendering();

  for (var key in computed) {
    var userDef = computed[key];
    var getter = typeof userDef === 'function' ? userDef : userDef.get;
    if (getter == null) {
      warn(
        ("Getter is missing for computed property \"" + key + "\"."),
        vm
      );
    }

    if (!isSSR) {
      // create internal watcher for the computed property.
      watchers[key] = new Watcher(
        vm,
        getter || noop,
        noop,
        computedWatcherOptions
      );
    }

    // component-defined computed properties are already defined on the
    // component prototype. We only need to define computed properties defined
    // at instantiation here.
    if (!(key in vm)) {
      defineComputed(vm, key, userDef);
    } else {
      if (key in vm.$data) {
        warn(("The computed property \"" + key + "\" is already defined in data."), vm);
      } else if (vm.$options.props && key in vm.$options.props) {
        warn(("The computed property \"" + key + "\" is already defined as a prop."), vm);
      }
    }
  }
}
```

```
function defineComputed (
  target,
  key,
  userDef
) {
  var shouldCache = !isServerRendering();
  if (typeof userDef === 'function') {
    sharedPropertyDefinition.get = shouldCache
      ? createComputedGetter(key)
      : createGetterInvoker(userDef);
    sharedPropertyDefinition.set = noop;
  } else {
    sharedPropertyDefinition.get = userDef.get
      ? shouldCache && userDef.cache !== false
        ? createComputedGetter(key)
        : createGetterInvoker(userDef.get)
      : noop;
    sharedPropertyDefinition.set = userDef.set || noop;
  }
  if (sharedPropertyDefinition.set === noop) {
    sharedPropertyDefinition.set = function () {
      warn(
        ("Computed property \"" + key + "\" was assigned to but it has no setter."),
        this
      );
    };
  }
  Object.defineProperty(target, key, sharedPropertyDefinition);
}

function createComputedGetter (key) {
  return function computedGetter () {
    var watcher = this._computedWatchers && this._computedWatchers[key];
    if (watcher) {
      if (watcher.dirty) {
        watcher.evaluate();
      }
      if (Dep.target) {
        watcher.depend();
      }
      return watcher.value
    }
  }
}

function createGetterInvoker(fn) {
  return function computedGetter () {
    return fn.call(this, this)
  }
}

function initMethods (vm, methods) {
  var props = vm.$options.props;
  for (var key in methods) {
    {
      if (typeof methods[key] !== 'function') {
        warn(
          "Method \"" + key + "\" has type \"" + (typeof methods[key]) + "\" in the component
          "Did you reference the function correctly?",
          vm
        );
      }
```

```
      if (props && hasOwn(props, key)) {
        warn(
          ("Method \"" + key + "\" has already been defined as a prop."),
          vm
        );
      }
      if ((key in vm) && isReserved(key)) {
        warn(
          "Method \"" + key + "\" conflicts with an existing Vue instance method. " +
          "Avoid defining component methods that start with _ or $."
        );
      }
    }
    vm[key] = typeof methods[key] !== 'function' ? noop : bind(methods[key], vm);
  }
}

function initWatch (vm, watch) {
  for (var key in watch) {
    var handler = watch[key];
    if (Array.isArray(handler)) {
      for (var i = 0; i < handler.length; i++) {
        createWatcher(vm, key, handler[i]);
      }
    } else {
      createWatcher(vm, key, handler);
    }
  }
}

function createWatcher (
  vm,
  expOrFn,
  handler,
  options
) {
  if (isPlainObject(handler)) {
    options = handler;
    handler = handler.handler;
  }
  if (typeof handler === 'string') {
    handler = vm[handler];
  }
  return vm.$watch(expOrFn, handler, options)
}

function stateMixin (Vue) {
  // flow somehow has problems with directly declared definition object
  // when using Object.defineProperty, so we have to procedurally build up
  // the object here.
  var dataDef = {};
  dataDef.get = function () { return this._data };
  var propsDef = {};
  propsDef.get = function () { return this._props };
  {
    dataDef.set = function () {
      warn(
        'Avoid replacing instance root $data. ' +
        'Use nested data properties instead.',
        this
      );
    };
```

```
    propsDef.set = function () {
      warn("$props is readonly.", this);
    };
  }
  Object.defineProperty(Vue.prototype, '$data', dataDef);
  Object.defineProperty(Vue.prototype, '$props', propsDef);

  Vue.prototype.$set = set;
  Vue.prototype.$delete = del;

  Vue.prototype.$watch = function (
    expOrFn,
    cb,
    options
  ) {
    var vm = this;
    if (isPlainObject(cb)) {
      return createWatcher(vm, expOrFn, cb, options)
    }
    options = options || {};
    options.user = true;
    var watcher = new Watcher(vm, expOrFn, cb, options);
    if (options.immediate) {
      try {
        cb.call(vm, watcher.value);
      } catch (error) {
        handleError(error, vm, ("callback for immediate watcher \"" + (watcher.expression) +
      }
    }
    return function unwatchFn () {
      watcher.teardown();
    }
  };
}

/*  */

var uid$3 = 0;

function initMixin (Vue) {
  Vue.prototype._init = function (options) {
    var vm = this;
    // a uid
    vm._uid = uid$3++;

    var startTag, endTag;
    /* istanbul ignore if */
    if (config.performance && mark) {
      startTag = "vue-perf-start:" + (vm._uid);
      endTag = "vue-perf-end:" + (vm._uid);
      mark(startTag);
    }

    // a flag to avoid this being observed
    vm._isVue = true;
    // merge options
    if (options && options._isComponent) {
      // optimize internal component instantiation
      // since dynamic options merging is pretty slow, and none of the
      // internal component options needs special treatment.
      initInternalComponent(vm, options);
    } else {
```

```
      vm.$options = mergeOptions(
        resolveConstructorOptions(vm.constructor),
        options || {},
        vm
      );
    }
    /* istanbul ignore else */
    {
      initProxy(vm);
    }
    // expose real self
    vm._self = vm;
    initLifecycle(vm);
    initEvents(vm);
    initRender(vm);
    callHook(vm, 'beforeCreate');
    initInjections(vm); // resolve injections before data/props
    initState(vm);
    initProvide(vm); // resolve provide after data/props
    callHook(vm, 'created');

    /* istanbul ignore if */
    if (config.performance && mark) {
      vm._name = formatComponentName(vm, false);
      mark(endTag);
      measure(("vue " + (vm._name) + " init"), startTag, endTag);
    }

    if (vm.$options.el) {
      vm.$mount(vm.$options.el);
    }
  };
}

function initInternalComponent (vm, options) {
  var opts = vm.$options = Object.create(vm.constructor.options);
  // doing this because it's faster than dynamic enumeration.
  var parentVnode = options._parentVnode;
  opts.parent = options.parent;
  opts._parentVnode = parentVnode;

  var vnodeComponentOptions = parentVnode.componentOptions;
  opts.propsData = vnodeComponentOptions.propsData;
  opts._parentListeners = vnodeComponentOptions.listeners;
  opts._renderChildren = vnodeComponentOptions.children;
  opts._componentTag = vnodeComponentOptions.tag;

  if (options.render) {
    opts.render = options.render;
    opts.staticRenderFns = options.staticRenderFns;
  }
}

function resolveConstructorOptions (Ctor) {
  var options = Ctor.options;
  if (Ctor.super) {
    var superOptions = resolveConstructorOptions(Ctor.super);
    var cachedSuperOptions = Ctor.superOptions;
    if (superOptions !== cachedSuperOptions) {
      // super option changed,
      // need to resolve new options.
      Ctor.superOptions = superOptions;
```

```
      // check if there are any late-modified/attached options (#4976)
      var modifiedOptions = resolveModifiedOptions(Ctor);
      // update base extend options
      if (modifiedOptions) {
        extend(Ctor.extendOptions, modifiedOptions);
      }
      options = Ctor.options = mergeOptions(superOptions, Ctor.extendOptions);
      if (options.name) {
        options.components[options.name] = Ctor;
      }
    }
  }
  return options
}

function resolveModifiedOptions (Ctor) {
  var modified;
  var latest = Ctor.options;
  var sealed = Ctor.sealedOptions;
  for (var key in latest) {
    if (latest[key] !== sealed[key]) {
      if (!modified) { modified = {}; }
      modified[key] = latest[key];
    }
  }
  return modified
}

function Vue (options) {
  if (!(this instanceof Vue)
  ) {
    warn('Vue is a constructor and should be called with the `new` keyword');
  }
  this._init(options);
}

initMixin(Vue);
stateMixin(Vue);
eventsMixin(Vue);
lifecycleMixin(Vue);
renderMixin(Vue);

/*  */

function initUse (Vue) {
  Vue.use = function (plugin) {
    var installedPlugins = (this._installedPlugins || (this._installedPlugins = []));
    if (installedPlugins.indexOf(plugin) > -1) {
      return this
    }

    // additional parameters
    var args = toArray(arguments, 1);
    args.unshift(this);
    if (typeof plugin.install === 'function') {
      plugin.install.apply(plugin, args);
    } else if (typeof plugin === 'function') {
      plugin.apply(null, args);
    }
    installedPlugins.push(plugin);
    return this
  };
```

```
}

/*  */

function initMixin$1 (Vue) {
  Vue.mixin = function (mixin) {
    this.options = mergeOptions(this.options, mixin);
    return this
  };
}

/*  */

function initExtend (Vue) {
  /**
   * Each instance constructor, including Vue, has a unique
   * cid. This enables us to create wrapped "child
   * constructors" for prototypal inheritance and cache them.
   */
  Vue.cid = 0;
  var cid = 1;

  /**
   * Class inheritance
   */
  Vue.extend = function (extendOptions) {
    extendOptions = extendOptions || {};
    var Super = this;
    var SuperId = Super.cid;
    var cachedCtors = extendOptions._Ctor || (extendOptions._Ctor = {});
    if (cachedCtors[SuperId]) {
      return cachedCtors[SuperId]
    }

    var name = extendOptions.name || Super.options.name;
    if (name) {
      validateComponentName(name);
    }

    var Sub = function VueComponent (options) {
      this._init(options);
    };
    Sub.prototype = Object.create(Super.prototype);
    Sub.prototype.constructor = Sub;
    Sub.cid = cid++;
    Sub.options = mergeOptions(
      Super.options,
      extendOptions
    );
    Sub['super'] = Super;

    // For props and computed properties, we define the proxy getters on
    // the Vue instances at extension time, on the extended prototype. This
    // avoids Object.defineProperty calls for each instance created.
    if (Sub.options.props) {
      initProps$1(Sub);
    }
    if (Sub.options.computed) {
      initComputed$1(Sub);
    }

    // allow further extension/mixin/plugin usage
```

```
    Sub.extend = Super.extend;
    Sub.mixin = Super.mixin;
    Sub.use = Super.use;

    // create asset registers, so extended classes
    // can have their private assets too.
    ASSET_TYPES.forEach(function (type) {
      Sub[type] = Super[type];
    });
    // enable recursive self-lookup
    if (name) {
      Sub.options.components[name] = Sub;
    }

    // keep a reference to the super options at extension time.
    // later at instantiation we can check if Super's options have
    // been updated.
    Sub.superOptions = Super.options;
    Sub.extendOptions = extendOptions;
    Sub.sealedOptions = extend({}, Sub.options);

    // cache constructor
    cachedCtors[SuperId] = Sub;
    return Sub
  };
}

function initProps$1 (Comp) {
  var props = Comp.options.props;
  for (var key in props) {
    proxy(Comp.prototype, "_props", key);
  }
}

function initComputed$1 (Comp) {
  var computed = Comp.options.computed;
  for (var key in computed) {
    defineComputed(Comp.prototype, key, computed[key]);
  }
}

/*  */

function initAssetRegisters (Vue) {
  /**
   * Create asset registration methods.
   */
  ASSET_TYPES.forEach(function (type) {
    Vue[type] = function (
      id,
      definition
    ) {
      if (!definition) {
        return this.options[type + 's'][id]
      } else {
        /* istanbul ignore if */
        if (type === 'component') {
          validateComponentName(id);
        }
        if (type === 'component' && isPlainObject(definition)) {
          definition.name = definition.name || id;
          definition = this.options._base.extend(definition);
```

```
        }
        if (type === 'directive' && typeof definition === 'function') {
          definition = { bind: definition, update: definition };
        }
        this.options[type + 's'][id] = definition;
        return definition
      }
    };
  });
}

/*  */



function getComponentName (opts) {
  return opts && (opts.Ctor.options.name || opts.tag)
}

function matches (pattern, name) {
  if (Array.isArray(pattern)) {
    return pattern.indexOf(name) > -1
  } else if (typeof pattern === 'string') {
    return pattern.split(',').indexOf(name) > -1
  } else if (isRegExp(pattern)) {
    return pattern.test(name)
  }
  /* istanbul ignore next */
  return false
}

function pruneCache (keepAliveInstance, filter) {
  var cache = keepAliveInstance.cache;
  var keys = keepAliveInstance.keys;
  var _vnode = keepAliveInstance._vnode;
  for (var key in cache) {
    var cachedNode = cache[key];
    if (cachedNode) {
      var name = getComponentName(cachedNode.componentOptions);
      if (name && !filter(name)) {
        pruneCacheEntry(cache, key, keys, _vnode);
      }
    }
  }
}

function pruneCacheEntry (
  cache,
  key,
  keys,
  current
) {
  var cached$$1 = cache[key];
  if (cached$$1 && (!current || cached$$1.tag !== current.tag)) {
    cached$$1.componentInstance.$destroy();
  }
  cache[key] = null;
  remove(keys, key);
}

var patternTypes = [String, RegExp, Array];
```

```
var KeepAlive = {
  name: 'keep-alive',
  abstract: true,

  props: {
    include: patternTypes,
    exclude: patternTypes,
    max: [String, Number]
  },

  created: function created () {
    this.cache = Object.create(null);
    this.keys = [];
  },

  destroyed: function destroyed () {
    for (var key in this.cache) {
      pruneCacheEntry(this.cache, key, this.keys);
    }
  },

  mounted: function mounted () {
    var this$1 = this;

    this.$watch('include', function (val) {
      pruneCache(this$1, function (name) { return matches(val, name); });
    });
    this.$watch('exclude', function (val) {
      pruneCache(this$1, function (name) { return !matches(val, name); });
    });
  },

  render: function render () {
    var slot = this.$slots.default;
    var vnode = getFirstComponentChild(slot);
    var componentOptions = vnode && vnode.componentOptions;
    if (componentOptions) {
      // check pattern
      var name = getComponentName(componentOptions);
      var ref = this;
      var include = ref.include;
      var exclude = ref.exclude;
      if (
        // not included
        (include && (!name || !matches(include, name))) ||
        // excluded
        (exclude && name && matches(exclude, name))
      ) {
        return vnode
      }

      var ref$1 = this;
      var cache = ref$1.cache;
      var keys = ref$1.keys;
      var key = vnode.key == null
        // same constructor may get registered as different local components
        // so cid alone is not enough (#3269)
        ? componentOptions.Ctor.cid + (componentOptions.tag ? ("::" + (componentOptions.tag)) : '')
        : vnode.key;
      if (cache[key]) {
        vnode.componentInstance = cache[key].componentInstance;
        // make current key freshest
```

```
        remove(keys, key);
        keys.push(key);
      } else {
        cache[key] = vnode;
        keys.push(key);
        // prune oldest entry
        if (this.max && keys.length > parseInt(this.max)) {
          pruneCacheEntry(cache, keys[0], keys, this._vnode);
        }
      }

      vnode.data.keepAlive = true;
    }
    return vnode || (slot && slot[0])
  }
};

var builtInComponents = {
  KeepAlive: KeepAlive
};

/*  */

function initGlobalAPI (Vue) {
  // config
  var configDef = {};
  configDef.get = function () { return config; };
  {
    configDef.set = function () {
      warn(
        'Do not replace the Vue.config object, set individual fields instead.'
      );
    };
  }
  Object.defineProperty(Vue, 'config', configDef);

  // exposed util methods.
  // NOTE: these are not considered part of the public API - avoid relying on
  // them unless you are aware of the risk.
  Vue.util = {
    warn: warn,
    extend: extend,
    mergeOptions: mergeOptions,
    defineReactive: defineReactive$$1
  };

  Vue.set = set;
  Vue.delete = del;
  Vue.nextTick = nextTick;

  // 2.6 explicit observable API
  Vue.observable = function (obj) {
    observe(obj);
    return obj
  };

  Vue.options = Object.create(null);
  ASSET_TYPES.forEach(function (type) {
    Vue.options[type + 's'] = Object.create(null);
  });

  // this is used to identify the "base" constructor to extend all plain-object
```

```
  // components with in Weex's multi-instance scenarios.
  Vue.options._base = Vue;

  extend(Vue.options.components, builtInComponents);

  initUse(Vue);
  initMixin$1(Vue);
  initExtend(Vue);
  initAssetRegisters(Vue);
}

initGlobalAPI(Vue);

Object.defineProperty(Vue.prototype, '$isServer', {
  get: isServerRendering
});

Object.defineProperty(Vue.prototype, '$ssrContext', {
  get: function get () {
    /* istanbul ignore next */
    return this.$vnode && this.$vnode.ssrContext
  }
});

// expose FunctionalRenderContext for ssr runtime helper installation
Object.defineProperty(Vue, 'FunctionalRenderContext', {
  value: FunctionalRenderContext
});

Vue.version = '2.6.10';

/*  */

// these are reserved for web because they are directly compiled away
// during template compilation
var isReservedAttr = makeMap('style,class');

// attributes that should be using props for binding
var acceptValue = makeMap('input,textarea,option,select,progress');
var mustUseProp = function (tag, type, attr) {
  return (
    (attr === 'value' && acceptValue(tag)) && type !== 'button' ||
    (attr === 'selected' && tag === 'option') ||
    (attr === 'checked' && tag === 'input') ||
    (attr === 'muted' && tag === 'video')
  )
};

var isEnumeratedAttr = makeMap('contenteditable,draggable,spellcheck');

var isValidContentEditableValue = makeMap('events,caret,typing,plaintext-only');

var convertEnumeratedValue = function (key, value) {
  return isFalsyAttrValue(value) || value === 'false'
    ? 'false'
    // allow arbitrary string value for contenteditable
    : key === 'contenteditable' && isValidContentEditableValue(value)
      ? value
      : 'true'
};

var isBooleanAttr = makeMap(
```

```
  'allowfullscreen,async,autofocus,autoplay,checked,compact,controls,declare,' +
  'default,defaultchecked,defaultmuted,defaultselected,defer,disabled,' +
  'enabled,formnovalidate,hidden,indeterminate,inert,ismap,itemscope,loop,multiple,' +
  'muted,nohref,noresize,noshade,novalidate,nowrap,open,pauseonexit,readonly,' +
  'required,reversed,scoped,seamless,selected,sortable,translate,' +
  'truespeed,typemustmatch,visible'
);

var xlinkNS = 'http://www.w3.org/1999/xlink';

var isXlink = function (name) {
  return name.charAt(5) === ':' && name.slice(0, 5) === 'xlink'
};

var getXlinkProp = function (name) {
  return isXlink(name) ? name.slice(6, name.length) : ''
};

var isFalsyAttrValue = function (val) {
  return val == null || val === false
};

/*  */

function genClassForVnode (vnode) {
  var data = vnode.data;
  var parentNode = vnode;
  var childNode = vnode;
  while (isDef(childNode.componentInstance)) {
    childNode = childNode.componentInstance._vnode;
    if (childNode && childNode.data) {
      data = mergeClassData(childNode.data, data);
    }
  }
  while (isDef(parentNode = parentNode.parent)) {
    if (parentNode && parentNode.data) {
      data = mergeClassData(data, parentNode.data);
    }
  }
  return renderClass(data.staticClass, data.class)
}

function mergeClassData (child, parent) {
  return {
    staticClass: concat(child.staticClass, parent.staticClass),
    class: isDef(child.class)
      ? [child.class, parent.class]
      : parent.class
  }
}

function renderClass (
  staticClass,
  dynamicClass
) {
  if (isDef(staticClass) || isDef(dynamicClass)) {
    return concat(staticClass, stringifyClass(dynamicClass))
  }
  /* istanbul ignore next */
  return ''
}
```

```
function concat (a, b) {
  return a ? b ? (a + ' ' + b) : a : (b || '')
}

function stringifyClass (value) {
  if (Array.isArray(value)) {
    return stringifyArray(value)
  }
  if (isObject(value)) {
    return stringifyObject(value)
  }
  if (typeof value === 'string') {
    return value
  }
  /* istanbul ignore next */
  return ''
}

function stringifyArray (value) {
  var res = '';
  var stringified;
  for (var i = 0, l = value.length; i < l; i++) {
    if (isDef(stringified = stringifyClass(value[i])) && stringified !== '') {
      if (res) { res += ' '; }
      res += stringified;
    }
  }
  return res
}

function stringifyObject (value) {
  var res = '';
  for (var key in value) {
    if (value[key]) {
      if (res) { res += ' '; }
      res += key;
    }
  }
  return res
}

/*  */

var namespaceMap = {
  svg: 'http://www.w3.org/2000/svg',
  math: 'http://www.w3.org/1998/Math/MathML'
};

var isHTMLTag = makeMap(
  'html,body,base,head,link,meta,style,title,' +
  'address,article,aside,footer,header,h1,h2,h3,h4,h5,h6,hgroup,nav,section,' +
  'div,dd,dl,dt,figcaption,figure,picture,hr,img,li,main,ol,p,pre,ul,' +
  'a,b,abbr,bdi,bdo,br,cite,code,data,dfn,em,i,kbd,mark,q,rp,rt,rtc,ruby,' +
  's,samp,small,span,strong,sub,sup,time,u,var,wbr,area,audio,map,track,video,' +
  'embed,object,param,source,canvas,script,noscript,del,ins,' +
  'caption,col,colgroup,table,thead,tbody,td,th,tr,' +
  'button,datalist,fieldset,form,input,label,legend,meter,optgroup,option,' +
  'output,progress,select,textarea,' +
  'details,dialog,menu,menuitem,summary,' +
  'content,element,shadow,template,blockquote,iframe,tfoot'
);
```

```
// this map is intentionally selective, only covering SVG elements that may
// contain child elements.
var isSVG = makeMap(
  'svg,animate,circle,clippath,cursor,defs,desc,ellipse,filter,font-face,' +
  'foreignObject,g,glyph,image,line,marker,mask,missing-glyph,path,pattern,' +
  'polygon,polyline,rect,switch,symbol,text,textpath,tspan,use,view',
  true
);

var isPreTag = function (tag) { return tag === 'pre'; };

var isReservedTag = function (tag) {
  return isHTMLTag(tag) || isSVG(tag)
};

function getTagNamespace (tag) {
  if (isSVG(tag)) {
    return 'svg'
  }
  // basic support for MathML
  // note it doesn't support other MathML elements being component roots
  if (tag === 'math') {
    return 'math'
  }
}

var unknownElementCache = Object.create(null);
function isUnknownElement (tag) {
  /* istanbul ignore if */
  if (!inBrowser) {
    return true
  }
  if (isReservedTag(tag)) {
    return false
  }
  tag = tag.toLowerCase();
  /* istanbul ignore if */
  if (unknownElementCache[tag] != null) {
    return unknownElementCache[tag]
  }
  var el = document.createElement(tag);
  if (tag.indexOf('-') > -1) {
    // http://stackoverflow.com/a/28210364/1070244
    return (unknownElementCache[tag] = (
      el.constructor === window.HTMLUnknownElement ||
      el.constructor === window.HTMLElement
    ))
  } else {
    return (unknownElementCache[tag] = /HTMLUnknownElement/.test(el.toString()))
  }
}

var isTextInputType = makeMap('text,number,password,search,email,tel,url');

/*  */

/**
 * Query an element selector if it's not an element already.
 */
function query (el) {
  if (typeof el === 'string') {
    var selected = document.querySelector(el);
```

```
    if (!selected) {
      warn(
        'Cannot find element: ' + el
      );
      return document.createElement('div')
    }
    return selected
  } else {
    return el
  }
}

/*  */

function createElement$1 (tagName, vnode) {
  var elm = document.createElement(tagName);
  if (tagName !== 'select') {
    return elm
  }
  // false or null will remove the attribute but undefined will not
  if (vnode.data && vnode.data.attrs && vnode.data.attrs.multiple !== undefined) {
    elm.setAttribute('multiple', 'multiple');
  }
  return elm
}

function createElementNS (namespace, tagName) {
  return document.createElementNS(namespaceMap[namespace], tagName)
}

function createTextNode (text) {
  return document.createTextNode(text)
}

function createComment (text) {
  return document.createComment(text)
}

function insertBefore (parentNode, newNode, referenceNode) {
  parentNode.insertBefore(newNode, referenceNode);
}

function removeChild (node, child) {
  node.removeChild(child);
}

function appendChild (node, child) {
  node.appendChild(child);
}

function parentNode (node) {
  return node.parentNode
}

function nextSibling (node) {
  return node.nextSibling
}

function tagName (node) {
  return node.tagName
}
```

```
function setTextContent (node, text) {
  node.textContent = text;
}

function setStyleScope (node, scopeId) {
  node.setAttribute(scopeId, '');
}

var nodeOps = /*#__PURE__*/Object.freeze({
  createElement: createElement$1,
  createElementNS: createElementNS,
  createTextNode: createTextNode,
  createComment: createComment,
  insertBefore: insertBefore,
  removeChild: removeChild,
  appendChild: appendChild,
  parentNode: parentNode,
  nextSibling: nextSibling,
  tagName: tagName,
  setTextContent: setTextContent,
  setStyleScope: setStyleScope
});

/*  */

var ref = {
  create: function create (_, vnode) {
    registerRef(vnode);
  },
  update: function update (oldVnode, vnode) {
    if (oldVnode.data.ref !== vnode.data.ref) {
      registerRef(oldVnode, true);
      registerRef(vnode);
    }
  },
  destroy: function destroy (vnode) {
    registerRef(vnode, true);
  }
};

function registerRef (vnode, isRemoval) {
  var key = vnode.data.ref;
  if (!isDef(key)) { return }

  var vm = vnode.context;
  var ref = vnode.componentInstance || vnode.elm;
  var refs = vm.$refs;
  if (isRemoval) {
    if (Array.isArray(refs[key])) {
      remove(refs[key], ref);
    } else if (refs[key] === ref) {
      refs[key] = undefined;
    }
  } else {
    if (vnode.data.refInFor) {
      if (!Array.isArray(refs[key])) {
        refs[key] = [ref];
      } else if (refs[key].indexOf(ref) < 0) {
        // $flow-disable-line
        refs[key].push(ref);
      }
    } else {
```

```
      refs[key] = ref;
    }
  }
}

/**
 * Virtual DOM patching algorithm based on Snabbdom by
 * Simon Friis Vindum (@paldepind)
 * Licensed under the MIT License
 * https://github.com/paldepind/snabbdom/blob/master/LICENSE
 *
 * modified by Evan You (@yyx990803)
 *
 * Not type-checking this because this file is perf-critical and the cost
 * of making flow understand it is not worth it.
 */

var emptyNode = new VNode('', {}, []);

var hooks = ['create', 'activate', 'update', 'remove', 'destroy'];

function sameVnode (a, b) {
  return (
    a.key === b.key && (
      (
        a.tag === b.tag &&
        a.isComment === b.isComment &&
        isDef(a.data) === isDef(b.data) &&
        sameInputType(a, b)
      ) || (
        isTrue(a.isAsyncPlaceholder) &&
        a.asyncFactory === b.asyncFactory &&
        isUndef(b.asyncFactory.error)
      )
    )
  )
}

function sameInputType (a, b) {
  if (a.tag !== 'input') { return true }
  var i;
  var typeA = isDef(i = a.data) && isDef(i = i.attrs) && i.type;
  var typeB = isDef(i = b.data) && isDef(i = i.attrs) && i.type;
  return typeA === typeB || isTextInputType(typeA) && isTextInputType(typeB)
}

function createKeyToOldIdx (children, beginIdx, endIdx) {
  var i, key;
  var map = {};
  for (i = beginIdx; i <= endIdx; ++i) {
    key = children[i].key;
    if (isDef(key)) { map[key] = i; }
  }
  return map
}

function createPatchFunction (backend) {
  var i, j;
  var cbs = {};

  var modules = backend.modules;
  var nodeOps = backend.nodeOps;
```

```
for (i = 0; i < hooks.length; ++i) {
  cbs[hooks[i]] = [];
  for (j = 0; j < modules.length; ++j) {
    if (isDef(modules[j][hooks[i]])) {
      cbs[hooks[i]].push(modules[j][hooks[i]]);
    }
  }
}

function emptyNodeAt (elm) {
  return new VNode(nodeOps.tagName(elm).toLowerCase(), {}, [], undefined, elm)
}

function createRmCb (childElm, listeners) {
  function remove$$1 () {
    if (--remove$$1.listeners === 0) {
      removeNode(childElm);
    }
  }
  remove$$1.listeners = listeners;
  return remove$$1
}

function removeNode (el) {
  var parent = nodeOps.parentNode(el);
  // element may have already been removed due to v-html / v-text
  if (isDef(parent)) {
    nodeOps.removeChild(parent, el);
  }
}

function isUnknownElement$$1 (vnode, inVPre) {
  return (
    !inVPre &&
    !vnode.ns &&
    !(
      config.ignoredElements.length &&
      config.ignoredElements.some(function (ignore) {
        return isRegExp(ignore)
          ? ignore.test(vnode.tag)
          : ignore === vnode.tag
      })
    ) &&
    config.isUnknownElement(vnode.tag)
  )
}

var creatingElmInVPre = 0;

function createElm (
  vnode,
  insertedVnodeQueue,
  parentElm,
  refElm,
  nested,
  ownerArray,
  index
) {
  if (isDef(vnode.elm) && isDef(ownerArray)) {
    // This vnode was used in a previous render!
    // now it's used as a new node, overwriting its elm would cause
```

```
    // potential patch errors down the road when it's used as an insertion
    // reference node. Instead, we clone the node on-demand before creating
    // associated DOM element for it.
    vnode = ownerArray[index] = cloneVNode(vnode);
  }

  vnode.isRootInsert = !nested; // for transition enter check
  if (createComponent(vnode, insertedVnodeQueue, parentElm, refElm)) {
    return
  }

  var data = vnode.data;
  var children = vnode.children;
  var tag = vnode.tag;
  if (isDef(tag)) {
    {
      if (data && data.pre) {
        creatingElmInVPre++;
      }
      if (isUnknownElement$$1(vnode, creatingElmInVPre)) {
        warn(
          'Unknown custom element: <' + tag + '> - did you ' +
          'register the component correctly? For recursive components, ' +
          'make sure to provide the "name" option.',
          vnode.context
        );
      }
    }

    vnode.elm = vnode.ns
      ? nodeOps.createElementNS(vnode.ns, tag)
      : nodeOps.createElement(tag, vnode);
    setScope(vnode);

    /* istanbul ignore if */
    {
      createChildren(vnode, children, insertedVnodeQueue);
      if (isDef(data)) {
        invokeCreateHooks(vnode, insertedVnodeQueue);
      }
      insert(parentElm, vnode.elm, refElm);
    }

    if (data && data.pre) {
      creatingElmInVPre--;
    }
  } else if (isTrue(vnode.isComment)) {
    vnode.elm = nodeOps.createComment(vnode.text);
    insert(parentElm, vnode.elm, refElm);
  } else {
    vnode.elm = nodeOps.createTextNode(vnode.text);
    insert(parentElm, vnode.elm, refElm);
  }
}

function createComponent (vnode, insertedVnodeQueue, parentElm, refElm) {
  var i = vnode.data;
  if (isDef(i)) {
    var isReactivated = isDef(vnode.componentInstance) && i.keepAlive;
    if (isDef(i = i.hook) && isDef(i = i.init)) {
      i(vnode, false /* hydrating */);
    }
```

```
    // after calling the init hook, if the vnode is a child component
    // it should've created a child instance and mounted it. the child
    // component also has set the placeholder vnode's elm.
    // in that case we can just return the element and be done.
    if (isDef(vnode.componentInstance)) {
      initComponent(vnode, insertedVnodeQueue);
      insert(parentElm, vnode.elm, refElm);
      if (isTrue(isReactivated)) {
        reactivateComponent(vnode, insertedVnodeQueue, parentElm, refElm);
      }
      return true
    }
  }
}

function initComponent (vnode, insertedVnodeQueue) {
  if (isDef(vnode.data.pendingInsert)) {
    insertedVnodeQueue.push.apply(insertedVnodeQueue, vnode.data.pendingInsert);
    vnode.data.pendingInsert = null;
  }
  vnode.elm = vnode.componentInstance.$el;
  if (isPatchable(vnode)) {
    invokeCreateHooks(vnode, insertedVnodeQueue);
    setScope(vnode);
  } else {
    // empty component root.
    // skip all element-related modules except for ref (#3455)
    registerRef(vnode);
    // make sure to invoke the insert hook
    insertedVnodeQueue.push(vnode);
  }
}

function reactivateComponent (vnode, insertedVnodeQueue, parentElm, refElm) {
  var i;
  // hack for #4339: a reactivated component with inner transition
  // does not trigger because the inner node's created hooks are not called
  // again. It's not ideal to involve module-specific logic in here but
  // there doesn't seem to be a better way to do it.
  var innerNode = vnode;
  while (innerNode.componentInstance) {
    innerNode = innerNode.componentInstance._vnode;
    if (isDef(i = innerNode.data) && isDef(i = i.transition)) {
      for (i = 0; i < cbs.activate.length; ++i) {
        cbs.activate[i](emptyNode, innerNode);
      }
      insertedVnodeQueue.push(innerNode);
      break
    }
  }
  // unlike a newly created component,
  // a reactivated keep-alive component doesn't insert itself
  insert(parentElm, vnode.elm, refElm);
}

function insert (parent, elm, ref$$1) {
  if (isDef(parent)) {
    if (isDef(ref$$1)) {
      if (nodeOps.parentNode(ref$$1) === parent) {
        nodeOps.insertBefore(parent, elm, ref$$1);
      }
    } else {
```

```
      nodeOps.appendChild(parent, elm);
    }
  }
}

function createChildren (vnode, children, insertedVnodeQueue) {
  if (Array.isArray(children)) {
    {
      checkDuplicateKeys(children);
    }
    for (var i = 0; i < children.length; ++i) {
      createElm(children[i], insertedVnodeQueue, vnode.elm, null, true, children, i);
    }
  } else if (isPrimitive(vnode.text)) {
    nodeOps.appendChild(vnode.elm, nodeOps.createTextNode(String(vnode.text)));
  }
}

function isPatchable (vnode) {
  while (vnode.componentInstance) {
    vnode = vnode.componentInstance._vnode;
  }
  return isDef(vnode.tag)
}

function invokeCreateHooks (vnode, insertedVnodeQueue) {
  for (var i$1 = 0; i$1 < cbs.create.length; ++i$1) {
    cbs.create[i$1](emptyNode, vnode);
  }
  i = vnode.data.hook; // Reuse variable
  if (isDef(i)) {
    if (isDef(i.create)) { i.create(emptyNode, vnode); }
    if (isDef(i.insert)) { insertedVnodeQueue.push(vnode); }
  }
}

// set scope id attribute for scoped CSS.
// this is implemented as a special case to avoid the overhead
// of going through the normal attribute patching process.
function setScope (vnode) {
  var i;
  if (isDef(i = vnode.fnScopeId)) {
    nodeOps.setStyleScope(vnode.elm, i);
  } else {
    var ancestor = vnode;
    while (ancestor) {
      if (isDef(i = ancestor.context) && isDef(i = i.$options._scopeId)) {
        nodeOps.setStyleScope(vnode.elm, i);
      }
      ancestor = ancestor.parent;
    }
  }
  // for slot content they should also get the scopeId from the host instance.
  if (isDef(i = activeInstance) &&
    i !== vnode.context &&
    i !== vnode.fnContext &&
    isDef(i = i.$options._scopeId)
  ) {
    nodeOps.setStyleScope(vnode.elm, i);
  }
}
```

```
function addVnodes (parentElm, refElm, vnodes, startIdx, endIdx, insertedVnodeQueue) {
  for (; startIdx <= endIdx; ++startIdx) {
    createElm(vnodes[startIdx], insertedVnodeQueue, parentElm, refElm, false, vnodes, start
  }
}

function invokeDestroyHook (vnode) {
  var i, j;
  var data = vnode.data;
  if (isDef(data)) {
    if (isDef(i = data.hook) && isDef(i = i.destroy)) { i(vnode); }
    for (i = 0; i < cbs.destroy.length; ++i) { cbs.destroy[i](vnode); }
  }
  if (isDef(i = vnode.children)) {
    for (j = 0; j < vnode.children.length; ++j) {
      invokeDestroyHook(vnode.children[j]);
    }
  }
}

function removeVnodes (parentElm, vnodes, startIdx, endIdx) {
  for (; startIdx <= endIdx; ++startIdx) {
    var ch = vnodes[startIdx];
    if (isDef(ch)) {
      if (isDef(ch.tag)) {
        removeAndInvokeRemoveHook(ch);
        invokeDestroyHook(ch);
      } else { // Text node
        removeNode(ch.elm);
      }
    }
  }
}

function removeAndInvokeRemoveHook (vnode, rm) {
  if (isDef(rm) || isDef(vnode.data)) {
    var i;
    var listeners = cbs.remove.length + 1;
    if (isDef(rm)) {
      // we have a recursively passed down rm callback
      // increase the listeners count
      rm.listeners += listeners;
    } else {
      // directly removing
      rm = createRmCb(vnode.elm, listeners);
    }
    // recursively invoke hooks on child component root node
    if (isDef(i = vnode.componentInstance) && isDef(i = i._vnode) && isDef(i.data)) {
      removeAndInvokeRemoveHook(i, rm);
    }
    for (i = 0; i < cbs.remove.length; ++i) {
      cbs.remove[i](vnode, rm);
    }
    if (isDef(i = vnode.data.hook) && isDef(i = i.remove)) {
      i(vnode, rm);
    } else {
      rm();
    }
  } else {
    removeNode(vnode.elm);
  }
}
```

```
function updateChildren (parentElm, oldCh, newCh, insertedVnodeQueue, removeOnly) {
  var oldStartIdx = 0;
  var newStartIdx = 0;
  var oldEndIdx = oldCh.length - 1;
  var oldStartVnode = oldCh[0];
  var oldEndVnode = oldCh[oldEndIdx];
  var newEndIdx = newCh.length - 1;
  var newStartVnode = newCh[0];
  var newEndVnode = newCh[newEndIdx];
  var oldKeyToIdx, idxInOld, vnodeToMove, refElm;

  // removeOnly is a special flag used only by <transition-group>
  // to ensure removed elements stay in correct relative positions
  // during leaving transitions
  var canMove = !removeOnly;

  {
    checkDuplicateKeys(newCh);
  }

  while (oldStartIdx <= oldEndIdx && newStartIdx <= newEndIdx) {
    if (isUndef(oldStartVnode)) {
      oldStartVnode = oldCh[++oldStartIdx]; // Vnode has been moved left
    } else if (isUndef(oldEndVnode)) {
      oldEndVnode = oldCh[--oldEndIdx];
    } else if (sameVnode(oldStartVnode, newStartVnode)) {
      patchVnode(oldStartVnode, newStartVnode, insertedVnodeQueue, newCh, newStartIdx);
      oldStartVnode = oldCh[++oldStartIdx];
      newStartVnode = newCh[++newStartIdx];
    } else if (sameVnode(oldEndVnode, newEndVnode)) {
      patchVnode(oldEndVnode, newEndVnode, insertedVnodeQueue, newCh, newEndIdx);
      oldEndVnode = oldCh[--oldEndIdx];
      newEndVnode = newCh[--newEndIdx];
    } else if (sameVnode(oldStartVnode, newEndVnode)) { // Vnode moved right
      patchVnode(oldStartVnode, newEndVnode, insertedVnodeQueue, newCh, newEndIdx);
      canMove && nodeOps.insertBefore(parentElm, oldStartVnode.elm, nodeOps.nextSibling(old
      oldStartVnode = oldCh[++oldStartIdx];
      newEndVnode = newCh[--newEndIdx];
    } else if (sameVnode(oldEndVnode, newStartVnode)) { // Vnode moved left
      patchVnode(oldEndVnode, newStartVnode, insertedVnodeQueue, newCh, newStartIdx);
      canMove && nodeOps.insertBefore(parentElm, oldEndVnode.elm, oldStartVnode.elm);
      oldEndVnode = oldCh[--oldEndIdx];
      newStartVnode = newCh[++newStartIdx];
    } else {
      if (isUndef(oldKeyToIdx)) { oldKeyToIdx = createKeyToOldIdx(oldCh, oldStartIdx, oldEr
      idxInOld = isDef(newStartVnode.key)
        ? oldKeyToIdx[newStartVnode.key]
        : findIdxInOld(newStartVnode, oldCh, oldStartIdx, oldEndIdx);
      if (isUndef(idxInOld)) { // New element
        createElm(newStartVnode, insertedVnodeQueue, parentElm, oldStartVnode.elm, false, r
      } else {
        vnodeToMove = oldCh[idxInOld];
        if (sameVnode(vnodeToMove, newStartVnode)) {
          patchVnode(vnodeToMove, newStartVnode, insertedVnodeQueue, newCh, newStartIdx);
          oldCh[idxInOld] = undefined;
          canMove && nodeOps.insertBefore(parentElm, vnodeToMove.elm, oldStartVnode.elm);
        } else {
          // same key but different element. treat as new element
          createElm(newStartVnode, insertedVnodeQueue, parentElm, oldStartVnode.elm, false,
        }
      }
```

```
      newStartVnode = newCh[++newStartIdx];
    }
  }
  if (oldStartIdx > oldEndIdx) {
    refElm = isUndef(newCh[newEndIdx + 1]) ? null : newCh[newEndIdx + 1].elm;
    addVnodes(parentElm, refElm, newCh, newStartIdx, newEndIdx, insertedVnodeQueue);
  } else if (newStartIdx > newEndIdx) {
    removeVnodes(parentElm, oldCh, oldStartIdx, oldEndIdx);
  }
}

function checkDuplicateKeys (children) {
  var seenKeys = {};
  for (var i = 0; i < children.length; i++) {
    var vnode = children[i];
    var key = vnode.key;
    if (isDef(key)) {
      if (seenKeys[key]) {
        warn(
          ("Duplicate keys detected: '" + key + "'. This may cause an update error."),
          vnode.context
        );
      } else {
        seenKeys[key] = true;
      }
    }
  }
}

function findIdxInOld (node, oldCh, start, end) {
  for (var i = start; i < end; i++) {
    var c = oldCh[i];
    if (isDef(c) && sameVnode(node, c)) { return i }
  }
}

function patchVnode (
  oldVnode,
  vnode,
  insertedVnodeQueue,
  ownerArray,
  index,
  removeOnly
) {
  if (oldVnode === vnode) {
    return
  }

  if (isDef(vnode.elm) && isDef(ownerArray)) {
    // clone reused vnode
    vnode = ownerArray[index] = cloneVNode(vnode);
  }

  var elm = vnode.elm = oldVnode.elm;

  if (isTrue(oldVnode.isAsyncPlaceholder)) {
    if (isDef(vnode.asyncFactory.resolved)) {
      hydrate(oldVnode.elm, vnode, insertedVnodeQueue);
    } else {
      vnode.isAsyncPlaceholder = true;
    }
    return
```

```
  }

  // reuse element for static trees.
  // note we only do this if the vnode is cloned -
  // if the new node is not cloned it means the render functions have been
  // reset by the hot-reload-api and we need to do a proper re-render.
  if (isTrue(vnode.isStatic) &&
    isTrue(oldVnode.isStatic) &&
    vnode.key === oldVnode.key &&
    (isTrue(vnode.isCloned) || isTrue(vnode.isOnce))
  ) {
    vnode.componentInstance = oldVnode.componentInstance;
    return
  }

  var i;
  var data = vnode.data;
  if (isDef(data) && isDef(i = data.hook) && isDef(i = i.prepatch)) {
    i(oldVnode, vnode);
  }

  var oldCh = oldVnode.children;
  var ch = vnode.children;
  if (isDef(data) && isPatchable(vnode)) {
    for (i = 0; i < cbs.update.length; ++i) { cbs.update[i](oldVnode, vnode); }
    if (isDef(i = data.hook) && isDef(i = i.update)) { i(oldVnode, vnode); }
  }
  if (isUndef(vnode.text)) {
    if (isDef(oldCh) && isDef(ch)) {
      if (oldCh !== ch) { updateChildren(elm, oldCh, ch, insertedVnodeQueue, removeOnly); }
    } else if (isDef(ch)) {
      {
        checkDuplicateKeys(ch);
      }
      if (isDef(oldVnode.text)) { nodeOps.setTextContent(elm, ''); }
      addVnodes(elm, null, ch, 0, ch.length - 1, insertedVnodeQueue);
    } else if (isDef(oldCh)) {
      removeVnodes(elm, oldCh, 0, oldCh.length - 1);
    } else if (isDef(oldVnode.text)) {
      nodeOps.setTextContent(elm, '');
    }
  } else if (oldVnode.text !== vnode.text) {
    nodeOps.setTextContent(elm, vnode.text);
  }
  if (isDef(data)) {
    if (isDef(i = data.hook) && isDef(i = i.postpatch)) { i(oldVnode, vnode); }
  }
}

function invokeInsertHook (vnode, queue, initial) {
  // delay insert hooks for component root nodes, invoke them after the
  // element is really inserted
  if (isTrue(initial) && isDef(vnode.parent)) {
    vnode.parent.data.pendingInsert = queue;
  } else {
    for (var i = 0; i < queue.length; ++i) {
      queue[i].data.hook.insert(queue[i]);
    }
  }
}

var hydrationBailed = false;
```

```
// list of modules that can skip create hook during hydration because they
// are already rendered on the client or has no need for initialization
// Note: style is excluded because it relies on initial clone for future
// deep updates (#7063).
var isRenderedModule = makeMap('attrs,class,staticClass,staticStyle,key');

// Note: this is a browser-only function so we can assume elms are DOM nodes.
function hydrate (elm, vnode, insertedVnodeQueue, inVPre) {
  var i;
  var tag = vnode.tag;
  var data = vnode.data;
  var children = vnode.children;
  inVPre = inVPre || (data && data.pre);
  vnode.elm = elm;

  if (isTrue(vnode.isComment) && isDef(vnode.asyncFactory)) {
    vnode.isAsyncPlaceholder = true;
    return true
  }
  // assert node match
  {
    if (!assertNodeMatch(elm, vnode, inVPre)) {
      return false
    }
  }
  if (isDef(data)) {
    if (isDef(i = data.hook) && isDef(i = i.init)) { i(vnode, true /* hydrating */); }
    if (isDef(i = vnode.componentInstance)) {
      // child component. it should have hydrated its own tree.
      initComponent(vnode, insertedVnodeQueue);
      return true
    }
  }
  if (isDef(tag)) {
    if (isDef(children)) {
      // empty element, allow client to pick up and populate children
      if (!elm.hasChildNodes()) {
        createChildren(vnode, children, insertedVnodeQueue);
      } else {
        // v-html and domProps: innerHTML
        if (isDef(i = data) && isDef(i = i.domProps) && isDef(i = i.innerHTML)) {
          if (i !== elm.innerHTML) {
            /* istanbul ignore if */
            if (typeof console !== 'undefined' &&
              !hydrationBailed
            ) {
              hydrationBailed = true;
              console.warn('Parent: ', elm);
              console.warn('server innerHTML: ', i);
              console.warn('client innerHTML: ', elm.innerHTML);
            }
            return false
          }
        } else {
          // iterate and compare children lists
          var childrenMatch = true;
          var childNode = elm.firstChild;
          for (var i$1 = 0; i$1 < children.length; i$1++) {
            if (!childNode || !hydrate(childNode, children[i$1], insertedVnodeQueue, inVPre
              childrenMatch = false;
              break
            }
```

```
            childNode = childNode.nextSibling;
          }
          // if childNode is not null, it means the actual childNodes list is
          // longer than the virtual children list.
          if (!childrenMatch || childNode) {
            /* istanbul ignore if */
            if (typeof console !== 'undefined' &&
              !hydrationBailed
            ) {
              hydrationBailed = true;
              console.warn('Parent: ', elm);
              console.warn('Mismatching childNodes vs. VNodes: ', elm.childNodes, children)
            }
            return false
          }
        }
      }
    }
    if (isDef(data)) {
      var fullInvoke = false;
      for (var key in data) {
        if (!isRenderedModule(key)) {
          fullInvoke = true;
          invokeCreateHooks(vnode, insertedVnodeQueue);
          break
        }
      }
      if (!fullInvoke && data['class']) {
        // ensure collecting deps for deep class bindings for future updates
        traverse(data['class']);
      }
    }
  } else if (elm.data !== vnode.text) {
    elm.data = vnode.text;
  }
  return true
}

function assertNodeMatch (node, vnode, inVPre) {
  if (isDef(vnode.tag)) {
    return vnode.tag.indexOf('vue-component') === 0 || (
      !isUnknownElement$$1(vnode, inVPre) &&
      vnode.tag.toLowerCase() === (node.tagName && node.tagName.toLowerCase())
    )
  } else {
    return node.nodeType === (vnode.isComment ? 8 : 3)
  }
}

return function patch (oldVnode, vnode, hydrating, removeOnly) {
  if (isUndef(vnode)) {
    if (isDef(oldVnode)) { invokeDestroyHook(oldVnode); }
    return
  }

  var isInitialPatch = false;
  var insertedVnodeQueue = [];

  if (isUndef(oldVnode)) {
    // empty mount (likely as component), create new root element
    isInitialPatch = true;
    createElm(vnode, insertedVnodeQueue);
```

```
  } else {
    var isRealElement = isDef(oldVnode.nodeType);
    if (!isRealElement && sameVnode(oldVnode, vnode)) {
      // patch existing root node
      patchVnode(oldVnode, vnode, insertedVnodeQueue, null, null, removeOnly);
    } else {
      if (isRealElement) {
        // mounting to a real element
        // check if this is server-rendered content and if we can perform
        // a successful hydration.
        if (oldVnode.nodeType === 1 && oldVnode.hasAttribute(SSR_ATTR)) {
          oldVnode.removeAttribute(SSR_ATTR);
          hydrating = true;
        }
        if (isTrue(hydrating)) {
          if (hydrate(oldVnode, vnode, insertedVnodeQueue)) {
            invokeInsertHook(vnode, insertedVnodeQueue, true);
            return oldVnode
          } else {
            warn(
              'The client-side rendered virtual DOM tree is not matching ' +
              'server-rendered content. This is likely caused by incorrect ' +
              'HTML markup, for example nesting block-level elements inside ' +
              '<p>, or missing <tbody>. Bailing hydration and performing ' +
              'full client-side render.'
            );
          }
        }
        // either not server-rendered, or hydration failed.
        // create an empty node and replace it
        oldVnode = emptyNodeAt(oldVnode);
      }

      // replacing existing element
      var oldElm = oldVnode.elm;
      var parentElm = nodeOps.parentNode(oldElm);

      // create new node
      createElm(
        vnode,
        insertedVnodeQueue,
        // extremely rare edge case: do not insert if old element is in a
        // leaving transition. Only happens when combining transition +
        // keep-alive + HOCs. (#4590)
        oldElm._leaveCb ? null : parentElm,
        nodeOps.nextSibling(oldElm)
      );

      // update parent placeholder node element, recursively
      if (isDef(vnode.parent)) {
        var ancestor = vnode.parent;
        var patchable = isPatchable(vnode);
        while (ancestor) {
          for (var i = 0; i < cbs.destroy.length; ++i) {
            cbs.destroy[i](ancestor);
          }
          ancestor.elm = vnode.elm;
          if (patchable) {
            for (var i$1 = 0; i$1 < cbs.create.length; ++i$1) {
              cbs.create[i$1](emptyNode, ancestor);
            }
            // #6513
```

```
              // invoke insert hooks that may have been merged by create hooks.
              // e.g. for directives that uses the "inserted" hook.
              var insert = ancestor.data.hook.insert;
              if (insert.merged) {
                // start at index 1 to avoid re-invoking component mounted hook
                for (var i$2 = 1; i$2 < insert.fns.length; i$2++) {
                  insert.fns[i$2]();
                }
              }
            } else {
              registerRef(ancestor);
            }
            ancestor = ancestor.parent;
          }
        }

        // destroy old node
        if (isDef(parentElm)) {
          removeVnodes(parentElm, [oldVnode], 0, 0);
        } else if (isDef(oldVnode.tag)) {
          invokeDestroyHook(oldVnode);
        }
      }
    }

    invokeInsertHook(vnode, insertedVnodeQueue, isInitialPatch);
    return vnode.elm
  }
}

/*  */

var directives = {
  create: updateDirectives,
  update: updateDirectives,
  destroy: function unbindDirectives (vnode) {
    updateDirectives(vnode, emptyNode);
  }
};

function updateDirectives (oldVnode, vnode) {
  if (oldVnode.data.directives || vnode.data.directives) {
    _update(oldVnode, vnode);
  }
}

function _update (oldVnode, vnode) {
  var isCreate = oldVnode === emptyNode;
  var isDestroy = vnode === emptyNode;
  var oldDirs = normalizeDirectives$1(oldVnode.data.directives, oldVnode.context);
  var newDirs = normalizeDirectives$1(vnode.data.directives, vnode.context);

  var dirsWithInsert = [];
  var dirsWithPostpatch = [];

  var key, oldDir, dir;
  for (key in newDirs) {
    oldDir = oldDirs[key];
    dir = newDirs[key];
    if (!oldDir) {
      // new directive, bind
      callHook$1(dir, 'bind', vnode, oldVnode);
```

```
      if (dir.def && dir.def.inserted) {
        dirsWithInsert.push(dir);
      }
    } else {
      // existing directive, update
      dir.oldValue = oldDir.value;
      dir.oldArg = oldDir.arg;
      callHook$1(dir, 'update', vnode, oldVnode);
      if (dir.def && dir.def.componentUpdated) {
        dirsWithPostpatch.push(dir);
      }
    }
  }

  if (dirsWithInsert.length) {
    var callInsert = function () {
      for (var i = 0; i < dirsWithInsert.length; i++) {
        callHook$1(dirsWithInsert[i], 'inserted', vnode, oldVnode);
      }
    };
    if (isCreate) {
      mergeVNodeHook(vnode, 'insert', callInsert);
    } else {
      callInsert();
    }
  }

  if (dirsWithPostpatch.length) {
    mergeVNodeHook(vnode, 'postpatch', function () {
      for (var i = 0; i < dirsWithPostpatch.length; i++) {
        callHook$1(dirsWithPostpatch[i], 'componentUpdated', vnode, oldVnode);
      }
    });
  }

  if (!isCreate) {
    for (key in oldDirs) {
      if (!newDirs[key]) {
        // no longer present, unbind
        callHook$1(oldDirs[key], 'unbind', oldVnode, oldVnode, isDestroy);
      }
    }
  }
}

var emptyModifiers = Object.create(null);

function normalizeDirectives$1 (
  dirs,
  vm
) {
  var res = Object.create(null);
  if (!dirs) {
    // $flow-disable-line
    return res
  }
  var i, dir;
  for (i = 0; i < dirs.length; i++) {
    dir = dirs[i];
    if (!dir.modifiers) {
      // $flow-disable-line
      dir.modifiers = emptyModifiers;
```

```
    }
    res[getRawDirName(dir)] = dir;
    dir.def = resolveAsset(vm.$options, 'directives', dir.name, true);
  }
  // $flow-disable-line
  return res
}

function getRawDirName (dir) {
  return dir.rawName || ((dir.name) + "." + (Object.keys(dir.modifiers || {}).join('.')))
}

function callHook$1 (dir, hook, vnode, oldVnode, isDestroy) {
  var fn = dir.def && dir.def[hook];
  if (fn) {
    try {
      fn(vnode.elm, dir, vnode, oldVnode, isDestroy);
    } catch (e) {
      handleError(e, vnode.context, ("directive " + (dir.name) + " " + hook + " hook"));
    }
  }
}

var baseModules = [
  ref,
  directives
];

/*  */

function updateAttrs (oldVnode, vnode) {
  var opts = vnode.componentOptions;
  if (isDef(opts) && opts.Ctor.options.inheritAttrs === false) {
    return
  }
  if (isUndef(oldVnode.data.attrs) && isUndef(vnode.data.attrs)) {
    return
  }
  var key, cur, old;
  var elm = vnode.elm;
  var oldAttrs = oldVnode.data.attrs || {};
  var attrs = vnode.data.attrs || {};
  // clone observed objects, as the user probably wants to mutate it
  if (isDef(attrs.__ob__)) {
    attrs = vnode.data.attrs = extend({}, attrs);
  }

  for (key in attrs) {
    cur = attrs[key];
    old = oldAttrs[key];
    if (old !== cur) {
      setAttr(elm, key, cur);
    }
  }
  // #4391: in IE9, setting type can reset value for input[type=radio]
  // #6666: IE/Edge forces progress value down to 1 before setting a max
  /* istanbul ignore if */
  if ((isIE || isEdge) && attrs.value !== oldAttrs.value) {
    setAttr(elm, 'value', attrs.value);
  }
  for (key in oldAttrs) {
    if (isUndef(attrs[key])) {
```

```
      if (isXlink(key)) {
        elm.removeAttributeNS(xlinkNS, getXlinkProp(key));
      } else if (!isEnumeratedAttr(key)) {
        elm.removeAttribute(key);
      }
    }
  }
}

function setAttr (el, key, value) {
  if (el.tagName.indexOf('-') > -1) {
    baseSetAttr(el, key, value);
  } else if (isBooleanAttr(key)) {
    // set attribute for blank value
    // e.g. <option disabled>Select one</option>
    if (isFalsyAttrValue(value)) {
      el.removeAttribute(key);
    } else {
      // technically allowfullscreen is a boolean attribute for <iframe>,
      // but Flash expects a value of "true" when used on <embed> tag
      value = key === 'allowfullscreen' && el.tagName === 'EMBED'
        ? 'true'
        : key;
      el.setAttribute(key, value);
    }
  } else if (isEnumeratedAttr(key)) {
    el.setAttribute(key, convertEnumeratedValue(key, value));
  } else if (isXlink(key)) {
    if (isFalsyAttrValue(value)) {
      el.removeAttributeNS(xlinkNS, getXlinkProp(key));
    } else {
      el.setAttributeNS(xlinkNS, key, value);
    }
  } else {
    baseSetAttr(el, key, value);
  }
}

function baseSetAttr (el, key, value) {
  if (isFalsyAttrValue(value)) {
    el.removeAttribute(key);
  } else {
    // #7138: IE10 & 11 fires input event when setting placeholder on
    // <textarea>... block the first input event and remove the blocker
    // immediately.
    /* istanbul ignore if */
    if (
      isIE && !isIE9 &&
      el.tagName === 'TEXTAREA' &&
      key === 'placeholder' && value !== '' && !el.__ieph
    ) {
      var blocker = function (e) {
        e.stopImmediatePropagation();
        el.removeEventListener('input', blocker);
      };
      el.addEventListener('input', blocker);
      // $flow-disable-line
      el.__ieph = true; /* IE placeholder patched */
    }
    el.setAttribute(key, value);
  }
}
```

```
var attrs = {
  create: updateAttrs,
  update: updateAttrs
};

/*  */

function updateClass (oldVnode, vnode) {
  var el = vnode.elm;
  var data = vnode.data;
  var oldData = oldVnode.data;
  if (
    isUndef(data.staticClass) &&
    isUndef(data.class) && (
      isUndef(oldData) || (
        isUndef(oldData.staticClass) &&
        isUndef(oldData.class)
      )
    )
  ) {
    return
  }

  var cls = genClassForVnode(vnode);

  // handle transition classes
  var transitionClass = el._transitionClasses;
  if (isDef(transitionClass)) {
    cls = concat(cls, stringifyClass(transitionClass));
  }

  // set the class
  if (cls !== el._prevClass) {
    el.setAttribute('class', cls);
    el._prevClass = cls;
  }
}

var klass = {
  create: updateClass,
  update: updateClass
};

/*  */

var validDivisionCharRE = /[\w).+\-_$\]]/;

function parseFilters (exp) {
  var inSingle = false;
  var inDouble = false;
  var inTemplateString = false;
  var inRegex = false;
  var curly = 0;
  var square = 0;
  var paren = 0;
  var lastFilterIndex = 0;
  var c, prev, i, expression, filters;

  for (i = 0; i < exp.length; i++) {
    prev = c;
    c = exp.charCodeAt(i);
```

```
  if (inSingle) {
    if (c === 0x27 && prev !== 0x5C) { inSingle = false; }
  } else if (inDouble) {
    if (c === 0x22 && prev !== 0x5C) { inDouble = false; }
  } else if (inTemplateString) {
    if (c === 0x60 && prev !== 0x5C) { inTemplateString = false; }
  } else if (inRegex) {
    if (c === 0x2f && prev !== 0x5C) { inRegex = false; }
  } else if (
    c === 0x7C && // pipe
    exp.charCodeAt(i + 1) !== 0x7C &&
    exp.charCodeAt(i - 1) !== 0x7C &&
    !curly && !square && !paren
  ) {
    if (expression === undefined) {
      // first filter, end of expression
      lastFilterIndex = i + 1;
      expression = exp.slice(0, i).trim();
    } else {
      pushFilter();
    }
  } else {
    switch (c) {
      case 0x22: inDouble = true; break         // "
      case 0x27: inSingle = true; break         // '
      case 0x60: inTemplateString = true; break // `
      case 0x28: paren++; break                 // (
      case 0x29: paren--; break                 // )
      case 0x5B: square++; break                // [
      case 0x5D: square--; break                // ]
      case 0x7B: curly++; break                 // {
      case 0x7D: curly--; break                 // }
    }
    if (c === 0x2f) { // /
      var j = i - 1;
      var p = (void 0);
      // find first non-whitespace prev char
      for (; j >= 0; j--) {
        p = exp.charAt(j);
        if (p !== ' ') { break }
      }
      if (!p || !validDivisionCharRE.test(p)) {
        inRegex = true;
      }
    }
  }
}

if (expression === undefined) {
  expression = exp.slice(0, i).trim();
} else if (lastFilterIndex !== 0) {
  pushFilter();
}

function pushFilter () {
  (filters || (filters = [])).push(exp.slice(lastFilterIndex, i).trim());
  lastFilterIndex = i + 1;
}

if (filters) {
  for (i = 0; i < filters.length; i++) {
    expression = wrapFilter(expression, filters[i]);
```

```
    }
  }

  return expression
}

function wrapFilter (exp, filter) {
  var i = filter.indexOf('(');
  if (i < 0) {
    // _f: resolveFilter
    return ("_f(\"" + filter + "\")(" + exp + ")")
  } else {
    var name = filter.slice(0, i);
    var args = filter.slice(i + 1);
    return ("_f(\"" + name + "\")(" + exp + (args !== ')' ? ',' + args : args))
  }
}

/*  */



/* eslint-disable no-unused-vars */
function baseWarn (msg, range) {
  console.error(("[Vue compiler]: " + msg));
}
/* eslint-enable no-unused-vars */

function pluckModuleFunction (
  modules,
  key
) {
  return modules
    ? modules.map(function (m) { return m[key]; }).filter(function (_) { return _; })
    : []
}

function addProp (el, name, value, range, dynamic) {
  (el.props || (el.props = [])).push(rangeSetItem({ name: name, value: value, dynamic: dynami
  el.plain = false;
}

function addAttr (el, name, value, range, dynamic) {
  var attrs = dynamic
    ? (el.dynamicAttrs || (el.dynamicAttrs = []))
    : (el.attrs || (el.attrs = []));
  attrs.push(rangeSetItem({ name: name, value: value, dynamic: dynamic }, range));
  el.plain = false;
}

// add a raw attr (use this in preTransforms)
function addRawAttr (el, name, value, range) {
  el.attrsMap[name] = value;
  el.attrsList.push(rangeSetItem({ name: name, value: value }, range));
}

function addDirective (
  el,
  name,
  rawName,
  value,
  arg,
```

```
  isDynamicArg,
  modifiers,
  range
) {
  (el.directives || (el.directives = [])).push(rangeSetItem({
    name: name,
    rawName: rawName,
    value: value,
    arg: arg,
    isDynamicArg: isDynamicArg,
    modifiers: modifiers
  }, range));
  el.plain = false;
}

function prependModifierMarker (symbol, name, dynamic) {
  return dynamic
    ? ("_p(" + name + ",\"" + symbol + "\")")
    : symbol + name // mark the event as captured
}

function addHandler (
  el,
  name,
  value,
  modifiers,
  important,
  warn,
  range,
  dynamic
) {
  modifiers = modifiers || emptyObject;
  // warn prevent and passive modifier
  /* istanbul ignore if */
  if (
    warn &&
    modifiers.prevent && modifiers.passive
  ) {
    warn(
      'passive and prevent can\'t be used together. ' +
      'Passive handler can\'t prevent default event.',
      range
    );
  }

  // normalize click.right and click.middle since they don't actually fire
  // this is technically browser-specific, but at least for now browsers are
  // the only target envs that have right/middle clicks.
  if (modifiers.right) {
    if (dynamic) {
      name = "(" + name + ")==='click'?'contextmenu':(" + name + ")";
    } else if (name === 'click') {
      name = 'contextmenu';
      delete modifiers.right;
    }
  } else if (modifiers.middle) {
    if (dynamic) {
      name = "(" + name + ")==='click'?'mouseup':(" + name + ")";
    } else if (name === 'click') {
      name = 'mouseup';
    }
  }
```

```
  // check capture modifier
  if (modifiers.capture) {
    delete modifiers.capture;
    name = prependModifierMarker('!', name, dynamic);
  }
  if (modifiers.once) {
    delete modifiers.once;
    name = prependModifierMarker('~', name, dynamic);
  }
  /* istanbul ignore if */
  if (modifiers.passive) {
    delete modifiers.passive;
    name = prependModifierMarker('&', name, dynamic);
  }

  var events;
  if (modifiers.native) {
    delete modifiers.native;
    events = el.nativeEvents || (el.nativeEvents = {});
  } else {
    events = el.events || (el.events = {});
  }

  var newHandler = rangeSetItem({ value: value.trim(), dynamic: dynamic }, range);
  if (modifiers !== emptyObject) {
    newHandler.modifiers = modifiers;
  }

  var handlers = events[name];
  /* istanbul ignore if */
  if (Array.isArray(handlers)) {
    important ? handlers.unshift(newHandler) : handlers.push(newHandler);
  } else if (handlers) {
    events[name] = important ? [newHandler, handlers] : [handlers, newHandler];
  } else {
    events[name] = newHandler;
  }

  el.plain = false;
}

function getRawBindingAttr (
  el,
  name
) {
  return el.rawAttrsMap[':' + name] ||
    el.rawAttrsMap['v-bind:' + name] ||
    el.rawAttrsMap[name]
}

function getBindingAttr (
  el,
  name,
  getStatic
) {
  var dynamicValue =
    getAndRemoveAttr(el, ':' + name) ||
    getAndRemoveAttr(el, 'v-bind:' + name);
  if (dynamicValue != null) {
    return parseFilters(dynamicValue)
  } else if (getStatic !== false) {
```

```
    var staticValue = getAndRemoveAttr(el, name);
    if (staticValue != null) {
      return JSON.stringify(staticValue)
    }
  }
}

// note: this only removes the attr from the Array (attrsList) so that it
// doesn't get processed by processAttrs.
// By default it does NOT remove it from the map (attrsMap) because the map is
// needed during codegen.
function getAndRemoveAttr (
  el,
  name,
  removeFromMap
) {
  var val;
  if ((val = el.attrsMap[name]) != null) {
    var list = el.attrsList;
    for (var i = 0, l = list.length; i < l; i++) {
      if (list[i].name === name) {
        list.splice(i, 1);
        break
      }
    }
  }
  if (removeFromMap) {
    delete el.attrsMap[name];
  }
  return val
}

function getAndRemoveAttrByRegex (
  el,
  name
) {
  var list = el.attrsList;
  for (var i = 0, l = list.length; i < l; i++) {
    var attr = list[i];
    if (name.test(attr.name)) {
      list.splice(i, 1);
      return attr
    }
  }
}

function rangeSetItem (
  item,
  range
) {
  if (range) {
    if (range.start != null) {
      item.start = range.start;
    }
    if (range.end != null) {
      item.end = range.end;
    }
  }
  return item
}

/*  */
```

```
/**
 * Cross-platform code generation for component v-model
 */
function genComponentModel (
  el,
  value,
  modifiers
) {
  var ref = modifiers || {};
  var number = ref.number;
  var trim = ref.trim;

  var baseValueExpression = '$$v';
  var valueExpression = baseValueExpression;
  if (trim) {
    valueExpression =
      "(typeof " + baseValueExpression + " === 'string'" +
      "? " + baseValueExpression + ".trim()" +
      ": " + baseValueExpression + ")";
  }
  if (number) {
    valueExpression = "_n(" + valueExpression + ")";
  }
  var assignment = genAssignmentCode(value, valueExpression);

  el.model = {
    value: ("(" + value + ")"),
    expression: JSON.stringify(value),
    callback: ("function (" + baseValueExpression + ") {" + assignment + "}")
  };
}

/**
 * Cross-platform codegen helper for generating v-model value assignment code.
 */
function genAssignmentCode (
  value,
  assignment
) {
  var res = parseModel(value);
  if (res.key === null) {
    return (value + "=" + assignment)
  } else {
    return ("$set(" + (res.exp) + ", " + (res.key) + ", " + assignment + ")")
  }
}

/**
 * Parse a v-model expression into a base path and a final key segment.
 * Handles both dot-path and possible square brackets.
 *
 * Possible cases:
 *
 * - test
 * - test[key]
 * - test[test1[key]]
 * - test["a"][key]
 * - xxx.test[a[a].test1[key]]
 * - test.xxx.a["asa"][test1[key]]
 *
 */
```

```
var len, str, chr, index$1, expressionPos, expressionEndPos;



function parseModel (val) {
  // Fix https://github.com/vuejs/vue/pull/7730
  // allow v-model="obj.val " (trailing whitespace)
  val = val.trim();
  len = val.length;

  if (val.indexOf('[') < 0 || val.lastIndexOf(']') < len - 1) {
    index$1 = val.lastIndexOf('.');
    if (index$1 > -1) {
      return {
        exp: val.slice(0, index$1),
        key: '"' + val.slice(index$1 + 1) + '"'
      }
    } else {
      return {
        exp: val,
        key: null
      }
    }
  }

  str = val;
  index$1 = expressionPos = expressionEndPos = 0;

  while (!eof()) {
    chr = next();
    /* istanbul ignore if */
    if (isStringStart(chr)) {
      parseString(chr);
    } else if (chr === 0x5B) {
      parseBracket(chr);
    }
  }

  return {
    exp: val.slice(0, expressionPos),
    key: val.slice(expressionPos + 1, expressionEndPos)
  }
}

function next () {
  return str.charCodeAt(++index$1)
}

function eof () {
  return index$1 >= len
}

function isStringStart (chr) {
  return chr === 0x22 || chr === 0x27
}

function parseBracket (chr) {
  var inBracket = 1;
  expressionPos = index$1;
  while (!eof()) {
    chr = next();
```

```
    if (isStringStart(chr)) {
      parseString(chr);
      continue
    }
    if (chr === 0x5B) { inBracket++; }
    if (chr === 0x5D) { inBracket--; }
    if (inBracket === 0) {
      expressionEndPos = index$1;
      break
    }
  }
}

function parseString (chr) {
  var stringQuote = chr;
  while (!eof()) {
    chr = next();
    if (chr === stringQuote) {
      break
    }
  }
}

/*  */

var warn$1;

// in some cases, the event used has to be determined at runtime
// so we used some reserved tokens during compile.
var RANGE_TOKEN = '__r';
var CHECKBOX_RADIO_TOKEN = '__c';

function model (
  el,
  dir,
  _warn
) {
  warn$1 = _warn;
  var value = dir.value;
  var modifiers = dir.modifiers;
  var tag = el.tag;
  var type = el.attrsMap.type;

  {
    // inputs with type="file" are read only and setting the input's
    // value will throw an error.
    if (tag === 'input' && type === 'file') {
      warn$1(
        "<" + (el.tag) + " v-model=\"" + value + "\" type=\"file\">:\n" +
        "File inputs are read only. Use a v-on:change listener instead.",
        el.rawAttrsMap['v-model']
      );
    }
  }

  if (el.component) {
    genComponentModel(el, value, modifiers);
    // component v-model doesn't need extra runtime
    return false
  } else if (tag === 'select') {
    genSelect(el, value, modifiers);
  } else if (tag === 'input' && type === 'checkbox') {
```

```
    genCheckboxModel(el, value, modifiers);
  } else if (tag === 'input' && type === 'radio') {
    genRadioModel(el, value, modifiers);
  } else if (tag === 'input' || tag === 'textarea') {
    genDefaultModel(el, value, modifiers);
  } else if (!config.isReservedTag(tag)) {
    genComponentModel(el, value, modifiers);
    // component v-model doesn't need extra runtime
    return false
  } else {
    warn$1(
      "<" + (el.tag) + " v-model=\"" + value + "\">: " +
      "v-model is not supported on this element type. " +
      'If you are working with contenteditable, it\'s recommended to ' +
      'wrap a library dedicated for that purpose inside a custom component.',
      el.rawAttrsMap['v-model']
    );
  }

  // ensure runtime directive metadata
  return true
}

function genCheckboxModel (
  el,
  value,
  modifiers
) {
  var number = modifiers && modifiers.number;
  var valueBinding = getBindingAttr(el, 'value') || 'null';
  var trueValueBinding = getBindingAttr(el, 'true-value') || 'true';
  var falseValueBinding = getBindingAttr(el, 'false-value') || 'false';
  addProp(el, 'checked',
    "Array.isArray(" + value + ")" +
    "?_i(" + value + "," + valueBinding + ")>-1" + (
      trueValueBinding === 'true'
        ? (":(" + value + ")")
        : (":_q(" + value + "," + trueValueBinding + ")")
    )
  );
  addHandler(el, 'change',
    "var $$a=" + value + "," +
        '$$el=$event.target,' +
        "$$c=$$el.checked?(" + trueValueBinding + "):(" + falseValueBinding + ");" +
    'if(Array.isArray($$a)){' +
      "var $$v=" + (number ? '_n(' + valueBinding + ')' : valueBinding) + "," +
          '$$i=_i($$a,$$v);' +
      "if($$el.checked){$$i<0&&(" + (genAssignmentCode(value, '$$a.concat([$$v])')) + ")}" +
      "else{$$i>-1&&(" + (genAssignmentCode(value, '$$a.slice(0,$$i).concat($$a.slice($$i+1))
    "}else{" + (genAssignmentCode(value, '$$c')) + "}",
    null, true
  );
}

function genRadioModel (
  el,
  value,
  modifiers
) {
  var number = modifiers && modifiers.number;
  var valueBinding = getBindingAttr(el, 'value') || 'null';
  valueBinding = number ? ("_n(" + valueBinding + ")") : valueBinding;
```

```
  addProp(el, 'checked', ("_q(" + value + "," + valueBinding + ")"));
  addHandler(el, 'change', genAssignmentCode(value, valueBinding), null, true);
}

function genSelect (
  el,
  value,
  modifiers
) {
  var number = modifiers && modifiers.number;
  var selectedVal = "Array.prototype.filter" +
    ".call($event.target.options,function(o){return o.selected})" +
    ".map(function(o){var val = \"_value\" in o ? o._value : o.value;" +
    "return " + (number ? '_n(val)' : 'val') + "})";

  var assignment = '$event.target.multiple ? $$selectedVal : $$selectedVal[0]';
  var code = "var $$selectedVal = " + selectedVal + ";";
  code = code + " " + (genAssignmentCode(value, assignment));
  addHandler(el, 'change', code, null, true);
}

function genDefaultModel (
  el,
  value,
  modifiers
) {
  var type = el.attrsMap.type;

  // warn if v-bind:value conflicts with v-model
  // except for inputs with v-bind:type
  {
    var value$1 = el.attrsMap['v-bind:value'] || el.attrsMap[':value'];
    var typeBinding = el.attrsMap['v-bind:type'] || el.attrsMap[':type'];
    if (value$1 && !typeBinding) {
      var binding = el.attrsMap['v-bind:value'] ? 'v-bind:value' : ':value';
      warn$1(
        binding + "=\"" + value$1 + "\" conflicts with v-model on the same element " +
        'because the latter already expands to a value binding internally',
        el.rawAttrsMap[binding]
      );
    }
  }

  var ref = modifiers || {};
  var lazy = ref.lazy;
  var number = ref.number;
  var trim = ref.trim;
  var needCompositionGuard = !lazy && type !== 'range';
  var event = lazy
    ? 'change'
    : type === 'range'
      ? RANGE_TOKEN
      : 'input';

  var valueExpression = '$event.target.value';
  if (trim) {
    valueExpression = "$event.target.value.trim()";
  }
  if (number) {
    valueExpression = "_n(" + valueExpression + ")";
  }
```

```
  var code = genAssignmentCode(value, valueExpression);
  if (needCompositionGuard) {
    code = "if($event.target.composing)return;" + code;
  }

  addProp(el, 'value', ("(" + value + ")"));
  addHandler(el, event, code, null, true);
  if (trim || number) {
    addHandler(el, 'blur', '$forceUpdate()');
  }
}

/*  */

// normalize v-model event tokens that can only be determined at runtime.
// it's important to place the event as the first in the array because
// the whole point is ensuring the v-model callback gets called before
// user-attached handlers.
function normalizeEvents (on) {
  /* istanbul ignore if */
  if (isDef(on[RANGE_TOKEN])) {
    // IE input[type=range] only supports `change` event
    var event = isIE ? 'change' : 'input';
    on[event] = [].concat(on[RANGE_TOKEN], on[event] || []);
    delete on[RANGE_TOKEN];
  }
  // This was originally intended to fix #4521 but no longer necessary
  // after 2.5. Keeping it for backwards compat with generated code from < 2.4
  /* istanbul ignore if */
  if (isDef(on[CHECKBOX_RADIO_TOKEN])) {
    on.change = [].concat(on[CHECKBOX_RADIO_TOKEN], on.change || []);
    delete on[CHECKBOX_RADIO_TOKEN];
  }
}

var target$1;

function createOnceHandler$1 (event, handler, capture) {
  var _target = target$1; // save current target element in closure
  return function onceHandler () {
    var res = handler.apply(null, arguments);
    if (res !== null) {
      remove$2(event, onceHandler, capture, _target);
    }
  }
}

// #9446: Firefox <= 53 (in particular, ESR 52) has incorrect Event.timeStamp
// implementation and does not fire microtasks in between event propagation, so
// safe to exclude.
var useMicrotaskFix = isUsingMicroTask && !(isFF && Number(isFF[1]) <= 53);

function add$1 (
  name,
  handler,
  capture,
  passive
) {
  // async edge case #6566: inner click event triggers patch, event handler
  // attached to outer element during patch, and triggered again. This
  // happens because browsers fire microtask ticks between event propagation.
  // the solution is simple: we save the timestamp when a handler is attached,
```

```
  // and the handler would only fire if the event passed to it was fired
  // AFTER it was attached.
  if (useMicrotaskFix) {
    var attachedTimestamp = currentFlushTimestamp;
    var original = handler;
    handler = original._wrapper = function (e) {
      if (
        // no bubbling, should always fire.
        // this is just a safety net in case event.timeStamp is unreliable in
        // certain weird environments...
        e.target === e.currentTarget ||
        // event is fired after handler attachment
        e.timeStamp >= attachedTimestamp ||
        // bail for environments that have buggy event.timeStamp implementations
        // #9462 iOS 9 bug: event.timeStamp is 0 after history.pushState
        // #9681 QtWebEngine event.timeStamp is negative value
        e.timeStamp <= 0 ||
        // #9448 bail if event is fired in another document in a multi-page
        // electron/nw.js app, since event.timeStamp will be using a different
        // starting reference
        e.target.ownerDocument !== document
      ) {
        return original.apply(this, arguments)
      }
    };
  }
  target$1.addEventListener(
    name,
    handler,
    supportsPassive
      ? { capture: capture, passive: passive }
      : capture
  );
}

function remove$2 (
  name,
  handler,
  capture,
  _target
) {
  (_target || target$1).removeEventListener(
    name,
    handler._wrapper || handler,
    capture
  );
}

function updateDOMListeners (oldVnode, vnode) {
  if (isUndef(oldVnode.data.on) && isUndef(vnode.data.on)) {
    return
  }
  var on = vnode.data.on || {};
  var oldOn = oldVnode.data.on || {};
  target$1 = vnode.elm;
  normalizeEvents(on);
  updateListeners(on, oldOn, add$1, remove$2, createOnceHandler$1, vnode.context);
  target$1 = undefined;
}

var events = {
  create: updateDOMListeners,
```

```
  update: updateDOMListeners
};

/*  */

var svgContainer;

function updateDOMProps (oldVnode, vnode) {
  if (isUndef(oldVnode.data.domProps) && isUndef(vnode.data.domProps)) {
    return
  }
  var key, cur;
  var elm = vnode.elm;
  var oldProps = oldVnode.data.domProps || {};
  var props = vnode.data.domProps || {};
  // clone observed objects, as the user probably wants to mutate it
  if (isDef(props.__ob__)) {
    props = vnode.data.domProps = extend({}, props);
  }

  for (key in oldProps) {
    if (!(key in props)) {
      elm[key] = '';
    }
  }

  for (key in props) {
    cur = props[key];
    // ignore children if the node has textContent or innerHTML,
    // as these will throw away existing DOM nodes and cause removal errors
    // on subsequent patches (#3360)
    if (key === 'textContent' || key === 'innerHTML') {
      if (vnode.children) { vnode.children.length = 0; }
      if (cur === oldProps[key]) { continue }
      // #6601 work around Chrome version <= 55 bug where single textNode
      // replaced by innerHTML/textContent retains its parentNode property
      if (elm.childNodes.length === 1) {
        elm.removeChild(elm.childNodes[0]);
      }
    }

    if (key === 'value' && elm.tagName !== 'PROGRESS') {
      // store value as _value as well since
      // non-string values will be stringified
      elm._value = cur;
      // avoid resetting cursor position when value is the same
      var strCur = isUndef(cur) ? '' : String(cur);
      if (shouldUpdateValue(elm, strCur)) {
        elm.value = strCur;
      }
    } else if (key === 'innerHTML' && isSVG(elm.tagName) && isUndef(elm.innerHTML)) {
      // IE doesn't support innerHTML for SVG elements
      svgContainer = svgContainer || document.createElement('div');
      svgContainer.innerHTML = "<svg>" + cur + "</svg>";
      var svg = svgContainer.firstChild;
      while (elm.firstChild) {
        elm.removeChild(elm.firstChild);
      }
      while (svg.firstChild) {
        elm.appendChild(svg.firstChild);
      }
    } else if (
```

```
      // skip the update if old and new VDOM state is the same.
      // `value` is handled separately because the DOM value may be temporarily
      // out of sync with VDOM state due to focus, composition and modifiers.
      // This  #4521 by skipping the unnecesarry `checked` update.
      cur !== oldProps[key]
    ) {
      // some property updates can throw
      // e.g. `value` on <progress> w/ non-finite value
      try {
        elm[key] = cur;
      } catch (e) {}
    }
  }
}

// check platforms/web/util/attrs.js acceptValue


function shouldUpdateValue (elm, checkVal) {
  return (!elm.composing && (
    elm.tagName === 'OPTION' ||
    isNotInFocusAndDirty(elm, checkVal) ||
    isDirtyWithModifiers(elm, checkVal)
  ))
}

function isNotInFocusAndDirty (elm, checkVal) {
  // return true when textbox (.number and .trim) loses focus and its value is
  // not equal to the updated value
  var notInFocus = true;
  // #6157
  // work around IE bug when accessing document.activeElement in an iframe
  try { notInFocus = document.activeElement !== elm; } catch (e) {}
  return notInFocus && elm.value !== checkVal
}

function isDirtyWithModifiers (elm, newVal) {
  var value = elm.value;
  var modifiers = elm._vModifiers; // injected by v-model runtime
  if (isDef(modifiers)) {
    if (modifiers.number) {
      return toNumber(value) !== toNumber(newVal)
    }
    if (modifiers.trim) {
      return value.trim() !== newVal.trim()
    }
  }
  return value !== newVal
}

var domProps = {
  create: updateDOMProps,
  update: updateDOMProps
};

/*  */

var parseStyleText = cached(function (cssText) {
  var res = {};
  var listDelimiter = /;(?![^(]*\))/g;
  var propertyDelimiter = /:(.+)/;
  cssText.split(listDelimiter).forEach(function (item) {
```

```
    if (item) {
      var tmp = item.split(propertyDelimiter);
      tmp.length > 1 && (res[tmp[0].trim()] = tmp[1].trim());
    }
  });
  return res
});

// merge static and dynamic style data on the same vnode
function normalizeStyleData (data) {
  var style = normalizeStyleBinding(data.style);
  // static style is pre-processed into an object during compilation
  // and is always a fresh object, so it's safe to merge into it
  return data.staticStyle
    ? extend(data.staticStyle, style)
    : style
}

// normalize possible array / string values into Object
function normalizeStyleBinding (bindingStyle) {
  if (Array.isArray(bindingStyle)) {
    return toObject(bindingStyle)
  }
  if (typeof bindingStyle === 'string') {
    return parseStyleText(bindingStyle)
  }
  return bindingStyle
}

/**
 * parent component style should be after child's
 * so that parent component's style could override it
 */
function getStyle (vnode, checkChild) {
  var res = {};
  var styleData;

  if (checkChild) {
    var childNode = vnode;
    while (childNode.componentInstance) {
      childNode = childNode.componentInstance._vnode;
      if (
        childNode && childNode.data &&
        (styleData = normalizeStyleData(childNode.data))
      ) {
        extend(res, styleData);
      }
    }
  }

  if ((styleData = normalizeStyleData(vnode.data))) {
    extend(res, styleData);
  }

  var parentNode = vnode;
  while ((parentNode = parentNode.parent)) {
    if (parentNode.data && (styleData = normalizeStyleData(parentNode.data))) {
      extend(res, styleData);
    }
  }
  return res
}
```

```
/*  */

var cssVarRE = /^--/;
var importantRE = /\s*!important$/;
var setProp = function (el, name, val) {
  /* istanbul ignore if */
  if (cssVarRE.test(name)) {
    el.style.setProperty(name, val);
  } else if (importantRE.test(val)) {
    el.style.setProperty(hyphenate(name), val.replace(importantRE, ''), 'important');
  } else {
    var normalizedName = normalize(name);
    if (Array.isArray(val)) {
      // Support values array created by autoprefixer, e.g.
      // {display: ["-webkit-box", "-ms-flexbox", "flex"]}
      // Set them one by one, and the browser will only set those it can recognize
      for (var i = 0, len = val.length; i < len; i++) {
        el.style[normalizedName] = val[i];
      }
    } else {
      el.style[normalizedName] = val;
    }
  }
};

var vendorNames = ['Webkit', 'Moz', 'ms'];

var emptyStyle;
var normalize = cached(function (prop) {
  emptyStyle = emptyStyle || document.createElement('div').style;
  prop = camelize(prop);
  if (prop !== 'filter' && (prop in emptyStyle)) {
    return prop
  }
  var capName = prop.charAt(0).toUpperCase() + prop.slice(1);
  for (var i = 0; i < vendorNames.length; i++) {
    var name = vendorNames[i] + capName;
    if (name in emptyStyle) {
      return name
    }
  }
});

function updateStyle (oldVnode, vnode) {
  var data = vnode.data;
  var oldData = oldVnode.data;

  if (isUndef(data.staticStyle) && isUndef(data.style) &&
    isUndef(oldData.staticStyle) && isUndef(oldData.style)
  ) {
    return
  }

  var cur, name;
  var el = vnode.elm;
  var oldStaticStyle = oldData.staticStyle;
  var oldStyleBinding = oldData.normalizedStyle || oldData.style || {};

  // if static style exists, stylebinding already merged into it when doing normalizeStyleDat
  var oldStyle = oldStaticStyle || oldStyleBinding;
```

```
  var style = normalizeStyleBinding(vnode.data.style) || {};

  // store normalized style under a different key for next diff
  // make sure to clone it if it's reactive, since the user likely wants
  // to mutate it.
  vnode.data.normalizedStyle = isDef(style.__ob__)
    ? extend({}, style)
    : style;

  var newStyle = getStyle(vnode, true);

  for (name in oldStyle) {
    if (isUndef(newStyle[name])) {
      setProp(el, name, '');
    }
  }
  for (name in newStyle) {
    cur = newStyle[name];
    if (cur !== oldStyle[name]) {
      // ie9 setting to null has no effect, must use empty string
      setProp(el, name, cur == null ? '' : cur);
    }
  }
}

var style = {
  create: updateStyle,
  update: updateStyle
};

/*  */

var whitespaceRE = /\s+/;

/**
 * Add class with compatibility for SVG since classList is not supported on
 * SVG elements in IE
 */
function addClass (el, cls) {
  /* istanbul ignore if */
  if (!cls || !(cls = cls.trim())) {
    return
  }

  /* istanbul ignore else */
  if (el.classList) {
    if (cls.indexOf(' ') > -1) {
      cls.split(whitespaceRE).forEach(function (c) { return el.classList.add(c); });
    } else {
      el.classList.add(cls);
    }
  } else {
    var cur = " " + (el.getAttribute('class') || '') + " ";
    if (cur.indexOf(' ' + cls + ' ') < 0) {
      el.setAttribute('class', (cur + cls).trim());
    }
  }
}

/**
 * Remove class with compatibility for SVG since classList is not supported on
 * SVG elements in IE
```

```
 */
function removeClass (el, cls) {
  /* istanbul ignore if */
  if (!cls || !(cls = cls.trim())) {
    return
  }

  /* istanbul ignore else */
  if (el.classList) {
    if (cls.indexOf(' ') > -1) {
      cls.split(whitespaceRE).forEach(function (c) { return el.classList.remove(c); });
    } else {
      el.classList.remove(cls);
    }
    if (!el.classList.length) {
      el.removeAttribute('class');
    }
  } else {
    var cur = " " + (el.getAttribute('class') || '') + " ";
    var tar = ' ' + cls + ' ';
    while (cur.indexOf(tar) >= 0) {
      cur = cur.replace(tar, ' ');
    }
    cur = cur.trim();
    if (cur) {
      el.setAttribute('class', cur);
    } else {
      el.removeAttribute('class');
    }
  }
}

/*  */

function resolveTransition (def$$1) {
  if (!def$$1) {
    return
  }
  /* istanbul ignore else */
  if (typeof def$$1 === 'object') {
    var res = {};
    if (def$$1.css !== false) {
      extend(res, autoCssTransition(def$$1.name || 'v'));
    }
    extend(res, def$$1);
    return res
  } else if (typeof def$$1 === 'string') {
    return autoCssTransition(def$$1)
  }
}

var autoCssTransition = cached(function (name) {
  return {
    enterClass: (name + "-enter"),
    enterToClass: (name + "-enter-to"),
    enterActiveClass: (name + "-enter-active"),
    leaveClass: (name + "-leave"),
    leaveToClass: (name + "-leave-to"),
    leaveActiveClass: (name + "-leave-active")
  }
});
```

```
var hasTransition = inBrowser && !isIE9;
var TRANSITION = 'transition';
var ANIMATION = 'animation';

// Transition property/event sniffing
var transitionProp = 'transition';
var transitionEndEvent = 'transitionend';
var animationProp = 'animation';
var animationEndEvent = 'animationend';
if (hasTransition) {
  /* istanbul ignore if */
  if (window.ontransitionend === undefined &&
    window.onwebkittransitionend !== undefined
  ) {
    transitionProp = 'WebkitTransition';
    transitionEndEvent = 'webkitTransitionEnd';
  }
  if (window.onanimationend === undefined &&
    window.onwebkitanimationend !== undefined
  ) {
    animationProp = 'WebkitAnimation';
    animationEndEvent = 'webkitAnimationEnd';
  }
}

// binding to window is necessary to make hot reload work in IE in strict mode
var raf = inBrowser
  ? window.requestAnimationFrame
    ? window.requestAnimationFrame.bind(window)
    : setTimeout
  : /* istanbul ignore next */ function (fn) { return fn(); };

function nextFrame (fn) {
  raf(function () {
    raf(fn);
  });
}

function addTransitionClass (el, cls) {
  var transitionClasses = el._transitionClasses || (el._transitionClasses = []);
  if (transitionClasses.indexOf(cls) < 0) {
    transitionClasses.push(cls);
    addClass(el, cls);
  }
}

function removeTransitionClass (el, cls) {
  if (el._transitionClasses) {
    remove(el._transitionClasses, cls);
  }
  removeClass(el, cls);
}

function whenTransitionEnds (
  el,
  expectedType,
  cb
) {
  var ref = getTransitionInfo(el, expectedType);
  var type = ref.type;
  var timeout = ref.timeout;
  var propCount = ref.propCount;
```

```
  if (!type) { return cb() }
  var event = type === TRANSITION ? transitionEndEvent : animationEndEvent;
  var ended = 0;
  var end = function () {
    el.removeEventListener(event, onEnd);
    cb();
  };
  var onEnd = function (e) {
    if (e.target === el) {
      if (++ended >= propCount) {
        end();
      }
    }
  };
  setTimeout(function () {
    if (ended < propCount) {
      end();
    }
  }, timeout + 1);
  el.addEventListener(event, onEnd);
}

var transformRE = /\b(transform|all)(,|$)/;

function getTransitionInfo (el, expectedType) {
  var styles = window.getComputedStyle(el);
  // JSDOM may return undefined for transition properties
  var transitionDelays = (styles[transitionProp + 'Delay'] || '').split(', ');
  var transitionDurations = (styles[transitionProp + 'Duration'] || '').split(', ');
  var transitionTimeout = getTimeout(transitionDelays, transitionDurations);
  var animationDelays = (styles[animationProp + 'Delay'] || '').split(', ');
  var animationDurations = (styles[animationProp + 'Duration'] || '').split(', ');
  var animationTimeout = getTimeout(animationDelays, animationDurations);

  var type;
  var timeout = 0;
  var propCount = 0;
  /* istanbul ignore if */
  if (expectedType === TRANSITION) {
    if (transitionTimeout > 0) {
      type = TRANSITION;
      timeout = transitionTimeout;
      propCount = transitionDurations.length;
    }
  } else if (expectedType === ANIMATION) {
    if (animationTimeout > 0) {
      type = ANIMATION;
      timeout = animationTimeout;
      propCount = animationDurations.length;
    }
  } else {
    timeout = Math.max(transitionTimeout, animationTimeout);
    type = timeout > 0
      ? transitionTimeout > animationTimeout
        ? TRANSITION
        : ANIMATION
      : null;
    propCount = type
      ? type === TRANSITION
        ? transitionDurations.length
        : animationDurations.length
      : 0;
```

```
  }
  var hasTransform =
    type === TRANSITION &&
    transformRE.test(styles[transitionProp + 'Property']);
  return {
    type: type,
    timeout: timeout,
    propCount: propCount,
    hasTransform: hasTransform
  }
}

function getTimeout (delays, durations) {
  /* istanbul ignore next */
  while (delays.length < durations.length) {
    delays = delays.concat(delays);
  }

  return Math.max.apply(null, durations.map(function (d, i) {
    return toMs(d) + toMs(delays[i])
  }))
}

// Old versions of Chromium (below 61.0.3163.100) formats floating pointer numbers
// in a locale-dependent way, using a comma instead of a dot.
// If comma is not replaced with a dot, the input will be rounded down (i.e. acting
// as a floor function) causing unexpected behaviors
function toMs (s) {
  return Number(s.slice(0, -1).replace(',', '.')) * 1000
}

/*  */

function enter (vnode, toggleDisplay) {
  var el = vnode.elm;

  // call leave callback now
  if (isDef(el._leaveCb)) {
    el._leaveCb.cancelled = true;
    el._leaveCb();
  }

  var data = resolveTransition(vnode.data.transition);
  if (isUndef(data)) {
    return
  }

  /* istanbul ignore if */
  if (isDef(el._enterCb) || el.nodeType !== 1) {
    return
  }

  var css = data.css;
  var type = data.type;
  var enterClass = data.enterClass;
  var enterToClass = data.enterToClass;
  var enterActiveClass = data.enterActiveClass;
  var appearClass = data.appearClass;
  var appearToClass = data.appearToClass;
  var appearActiveClass = data.appearActiveClass;
  var beforeEnter = data.beforeEnter;
  var enter = data.enter;
```

```
var afterEnter = data.afterEnter;
var enterCancelled = data.enterCancelled;
var beforeAppear = data.beforeAppear;
var appear = data.appear;
var afterAppear = data.afterAppear;
var appearCancelled = data.appearCancelled;
var duration = data.duration;

// activeInstance will always be the <transition> component managing this
// transition. One edge case to check is when the <transition> is placed
// as the root node of a child component. In that case we need to check
// <transition>'s parent for appear check.
var context = activeInstance;
var transitionNode = activeInstance.$vnode;
while (transitionNode && transitionNode.parent) {
  context = transitionNode.context;
  transitionNode = transitionNode.parent;
}

var isAppear = !context._isMounted || !vnode.isRootInsert;

if (isAppear && !appear && appear !== '') {
  return
}

var startClass = isAppear && appearClass
  ? appearClass
  : enterClass;
var activeClass = isAppear && appearActiveClass
  ? appearActiveClass
  : enterActiveClass;
var toClass = isAppear && appearToClass
  ? appearToClass
  : enterToClass;

var beforeEnterHook = isAppear
  ? (beforeAppear || beforeEnter)
  : beforeEnter;
var enterHook = isAppear
  ? (typeof appear === 'function' ? appear : enter)
  : enter;
var afterEnterHook = isAppear
  ? (afterAppear || afterEnter)
  : afterEnter;
var enterCancelledHook = isAppear
  ? (appearCancelled || enterCancelled)
  : enterCancelled;

var explicitEnterDuration = toNumber(
  isObject(duration)
    ? duration.enter
    : duration
);

if (explicitEnterDuration != null) {
  checkDuration(explicitEnterDuration, 'enter', vnode);
}

var expectsCSS = css !== false && !isIE9;
var userWantsControl = getHookArgumentsLength(enterHook);

var cb = el._enterCb = once(function () {
```

```
    if (expectsCSS) {
      removeTransitionClass(el, toClass);
      removeTransitionClass(el, activeClass);
    }
    if (cb.cancelled) {
      if (expectsCSS) {
        removeTransitionClass(el, startClass);
      }
      enterCancelledHook && enterCancelledHook(el);
    } else {
      afterEnterHook && afterEnterHook(el);
    }
    el._enterCb = null;
  });

  if (!vnode.data.show) {
    // remove pending leave element on enter by injecting an insert hook
    mergeVNodeHook(vnode, 'insert', function () {
      var parent = el.parentNode;
      var pendingNode = parent && parent._pending && parent._pending[vnode.key];
      if (pendingNode &&
        pendingNode.tag === vnode.tag &&
        pendingNode.elm._leaveCb
      ) {
        pendingNode.elm._leaveCb();
      }
      enterHook && enterHook(el, cb);
    });
  }

  // start enter transition
  beforeEnterHook && beforeEnterHook(el);
  if (expectsCSS) {
    addTransitionClass(el, startClass);
    addTransitionClass(el, activeClass);
    nextFrame(function () {
      removeTransitionClass(el, startClass);
      if (!cb.cancelled) {
        addTransitionClass(el, toClass);
        if (!userWantsControl) {
          if (isValidDuration(explicitEnterDuration)) {
            setTimeout(cb, explicitEnterDuration);
          } else {
            whenTransitionEnds(el, type, cb);
          }
        }
      }
    });
  }

  if (vnode.data.show) {
    toggleDisplay && toggleDisplay();
    enterHook && enterHook(el, cb);
  }

  if (!expectsCSS && !userWantsControl) {
    cb();
  }
}

function leave (vnode, rm) {
  var el = vnode.elm;
```

```
// call enter callback now
if (isDef(el._enterCb)) {
  el._enterCb.cancelled = true;
  el._enterCb();
}

var data = resolveTransition(vnode.data.transition);
if (isUndef(data) || el.nodeType !== 1) {
  return rm()
}

/* istanbul ignore if */
if (isDef(el._leaveCb)) {
  return
}

var css = data.css;
var type = data.type;
var leaveClass = data.leaveClass;
var leaveToClass = data.leaveToClass;
var leaveActiveClass = data.leaveActiveClass;
var beforeLeave = data.beforeLeave;
var leave = data.leave;
var afterLeave = data.afterLeave;
var leaveCancelled = data.leaveCancelled;
var delayLeave = data.delayLeave;
var duration = data.duration;

var expectsCSS = css !== false && !isIE9;
var userWantsControl = getHookArgumentsLength(leave);

var explicitLeaveDuration = toNumber(
  isObject(duration)
    ? duration.leave
    : duration
);

if (isDef(explicitLeaveDuration)) {
  checkDuration(explicitLeaveDuration, 'leave', vnode);
}

var cb = el._leaveCb = once(function () {
  if (el.parentNode && el.parentNode._pending) {
    el.parentNode._pending[vnode.key] = null;
  }
  if (expectsCSS) {
    removeTransitionClass(el, leaveToClass);
    removeTransitionClass(el, leaveActiveClass);
  }
  if (cb.cancelled) {
    if (expectsCSS) {
      removeTransitionClass(el, leaveClass);
    }
    leaveCancelled && leaveCancelled(el);
  } else {
    rm();
    afterLeave && afterLeave(el);
  }
  el._leaveCb = null;
});
```

```
  if (delayLeave) {
    delayLeave(performLeave);
  } else {
    performLeave();
  }

  function performLeave () {
    // the delayed leave may have already been cancelled
    if (cb.cancelled) {
      return
    }
    // record leaving element
    if (!vnode.data.show && el.parentNode) {
      (el.parentNode._pending || (el.parentNode._pending = {}))[(vnode.key)] = vnode;
    }
    beforeLeave && beforeLeave(el);
    if (expectsCSS) {
      addTransitionClass(el, leaveClass);
      addTransitionClass(el, leaveActiveClass);
      nextFrame(function () {
        removeTransitionClass(el, leaveClass);
        if (!cb.cancelled) {
          addTransitionClass(el, leaveToClass);
          if (!userWantsControl) {
            if (isValidDuration(explicitLeaveDuration)) {
              setTimeout(cb, explicitLeaveDuration);
            } else {
              whenTransitionEnds(el, type, cb);
            }
          }
        }
      });
    }
    leave && leave(el, cb);
    if (!expectsCSS && !userWantsControl) {
      cb();
    }
  }
}

// only used in dev mode
function checkDuration (val, name, vnode) {
  if (typeof val !== 'number') {
    warn(
      "<transition> explicit " + name + " duration is not a valid number - " +
      "got " + (JSON.stringify(val)) + ".",
      vnode.context
    );
  } else if (isNaN(val)) {
    warn(
      "<transition> explicit " + name + " duration is NaN - " +
      'the duration expression might be incorrect.',
      vnode.context
    );
  }
}

function isValidDuration (val) {
  return typeof val === 'number' && !isNaN(val)
}

/**
```

```
 * Normalize a transition hook's argument length. The hook may be:
 * - a merged hook (invoker) with the original in .fns
 * - a wrapped component method (check ._length)
 * - a plain function (.length)
 */
function getHookArgumentsLength (fn) {
  if (isUndef(fn)) {
    return false
  }
  var invokerFns = fn.fns;
  if (isDef(invokerFns)) {
    // invoker
    return getHookArgumentsLength(
      Array.isArray(invokerFns)
        ? invokerFns[0]
        : invokerFns
    )
  } else {
    return (fn._length || fn.length) > 1
  }
}

function _enter (_, vnode) {
  if (vnode.data.show !== true) {
    enter(vnode);
  }
}

var transition = inBrowser ? {
  create: _enter,
  activate: _enter,
  remove: function remove$$1 (vnode, rm) {
    /* istanbul ignore else */
    if (vnode.data.show !== true) {
      leave(vnode, rm);
    } else {
      rm();
    }
  }
} : {};

var platformModules = [
  attrs,
  klass,
  events,
  domProps,
  style,
  transition
];

/*  */

// the directive module should be applied last, after all
// built-in modules have been applied.
var modules = platformModules.concat(baseModules);

var patch = createPatchFunction({ nodeOps: nodeOps, modules: modules });

/**
 * Not type checking this file because flow doesn't like attaching
 * properties to Elements.
 */
```

```
/* istanbul ignore if */
if (isIE9) {
  // http://www.matts411.com/post/internet-explorer-9-oninput/
  document.addEventListener('selectionchange', function () {
    var el = document.activeElement;
    if (el && el.vmodel) {
      trigger(el, 'input');
    }
  });
}

var directive = {
  inserted: function inserted (el, binding, vnode, oldVnode) {
    if (vnode.tag === 'select') {
      // #6903
      if (oldVnode.elm && !oldVnode.elm._vOptions) {
        mergeVNodeHook(vnode, 'postpatch', function () {
          directive.componentUpdated(el, binding, vnode);
        });
      } else {
        setSelected(el, binding, vnode.context);
      }
      el._vOptions = [].map.call(el.options, getValue);
    } else if (vnode.tag === 'textarea' || isTextInputType(el.type)) {
      el._vModifiers = binding.modifiers;
      if (!binding.modifiers.lazy) {
        el.addEventListener('compositionstart', onCompositionStart);
        el.addEventListener('compositionend', onCompositionEnd);
        // Safari < 10.2 & UIWebView doesn't fire compositionend when
        // switching focus before confirming composition choice
        // this also fixes the issue where some browsers e.g. iOS Chrome
        // fires "change" instead of "input" on autocomplete.
        el.addEventListener('change', onCompositionEnd);
        /* istanbul ignore if */
        if (isIE9) {
          el.vmodel = true;
        }
      }
    }
  },

  componentUpdated: function componentUpdated (el, binding, vnode) {
    if (vnode.tag === 'select') {
      setSelected(el, binding, vnode.context);
      // in case the options rendered by v-for have changed,
      // it's possible that the value is out-of-sync with the rendered options.
      // detect such cases and filter out values that no longer has a matching
      // option in the DOM.
      var prevOptions = el._vOptions;
      var curOptions = el._vOptions = [].map.call(el.options, getValue);
      if (curOptions.some(function (o, i) { return !looseEqual(o, prevOptions[i]); })) {
        // trigger change event if
        // no matching option found for at least one value
        var needReset = el.multiple
          ? binding.value.some(function (v) { return hasNoMatchingOption(v, curOptions); })
          : binding.value !== binding.oldValue && hasNoMatchingOption(binding.value, curOptic
        if (needReset) {
          trigger(el, 'change');
        }
      }
    }
```

```
  }
};

function setSelected (el, binding, vm) {
  actuallySetSelected(el, binding, vm);
  /* istanbul ignore if */
  if (isIE || isEdge) {
    setTimeout(function () {
      actuallySetSelected(el, binding, vm);
    }, 0);
  }
}

function actuallySetSelected (el, binding, vm) {
  var value = binding.value;
  var isMultiple = el.multiple;
  if (isMultiple && !Array.isArray(value)) {
    warn(
      "<select multiple v-model=\"" + (binding.expression) + "\"> " +
      "expects an Array value for its binding, but got " + (Object.prototype.toString.call(va
      vm
    );
    return
  }
  var selected, option;
  for (var i = 0, l = el.options.length; i < l; i++) {
    option = el.options[i];
    if (isMultiple) {
      selected = looseIndexOf(value, getValue(option)) > -1;
      if (option.selected !== selected) {
        option.selected = selected;
      }
    } else {
      if (looseEqual(getValue(option), value)) {
        if (el.selectedIndex !== i) {
          el.selectedIndex = i;
        }
        return
      }
    }
  }
  if (!isMultiple) {
    el.selectedIndex = -1;
  }
}

function hasNoMatchingOption (value, options) {
  return options.every(function (o) { return !looseEqual(o, value); })
}

function getValue (option) {
  return '_value' in option
    ? option._value
    : option.value
}

function onCompositionStart (e) {
  e.target.composing = true;
}

function onCompositionEnd (e) {
  // prevent triggering an input event for no reason
```

```
  if (!e.target.composing) { return }
  e.target.composing = false;
  trigger(e.target, 'input');
}

function trigger (el, type) {
  var e = document.createEvent('HTMLEvents');
  e.initEvent(type, true, true);
  el.dispatchEvent(e);
}

/*  */

// recursively search for possible transition defined inside the component root
function locateNode (vnode) {
  return vnode.componentInstance && (!vnode.data || !vnode.data.transition)
    ? locateNode(vnode.componentInstance._vnode)
    : vnode
}

var show = {
  bind: function bind (el, ref, vnode) {
    var value = ref.value;

    vnode = locateNode(vnode);
    var transition$$1 = vnode.data && vnode.data.transition;
    var originalDisplay = el.__vOriginalDisplay =
      el.style.display === 'none' ? '' : el.style.display;
    if (value && transition$$1) {
      vnode.data.show = true;
      enter(vnode, function () {
        el.style.display = originalDisplay;
      });
    } else {
      el.style.display = value ? originalDisplay : 'none';
    }
  },

  update: function update (el, ref, vnode) {
    var value = ref.value;
    var oldValue = ref.oldValue;

    /* istanbul ignore if */
    if (!value === !oldValue) { return }
    vnode = locateNode(vnode);
    var transition$$1 = vnode.data && vnode.data.transition;
    if (transition$$1) {
      vnode.data.show = true;
      if (value) {
        enter(vnode, function () {
          el.style.display = el.__vOriginalDisplay;
        });
      } else {
        leave(vnode, function () {
          el.style.display = 'none';
        });
      }
    } else {
      el.style.display = value ? el.__vOriginalDisplay : 'none';
    }
  },
```

```
  unbind: function unbind (
    el,
    binding,
    vnode,
    oldVnode,
    isDestroy
  ) {
    if (!isDestroy) {
      el.style.display = el.__vOriginalDisplay;
    }
  }
};

var platformDirectives = {
  model: directive,
  show: show
};

/*  */

var transitionProps = {
  name: String,
  appear: Boolean,
  css: Boolean,
  mode: String,
  type: String,
  enterClass: String,
  leaveClass: String,
  enterToClass: String,
  leaveToClass: String,
  enterActiveClass: String,
  leaveActiveClass: String,
  appearClass: String,
  appearActiveClass: String,
  appearToClass: String,
  duration: [Number, String, Object]
};

// in case the child is also an abstract component, e.g. <keep-alive>
// we want to recursively retrieve the real component to be rendered
function getRealChild (vnode) {
  var compOptions = vnode && vnode.componentOptions;
  if (compOptions && compOptions.Ctor.options.abstract) {
    return getRealChild(getFirstComponentChild(compOptions.children))
  } else {
    return vnode
  }
}

function extractTransitionData (comp) {
  var data = {};
  var options = comp.$options;
  // props
  for (var key in options.propsData) {
    data[key] = comp[key];
  }
  // events.
  // extract listeners and pass them directly to the transition methods
  var listeners = options._parentListeners;
  for (var key$1 in listeners) {
    data[camelize(key$1)] = listeners[key$1];
  }
```

```
  return data
}

function placeholder (h, rawChild) {
  if (/\d-keep-alive$/.test(rawChild.tag)) {
    return h('keep-alive', {
      props: rawChild.componentOptions.propsData
    })
  }
}

function hasParentTransition (vnode) {
  while ((vnode = vnode.parent)) {
    if (vnode.data.transition) {
      return true
    }
  }
}

function isSameChild (child, oldChild) {
  return oldChild.key === child.key && oldChild.tag === child.tag
}

var isNotTextNode = function (c) { return c.tag || isAsyncPlaceholder(c); };

var isVShowDirective = function (d) { return d.name === 'show'; };

var Transition = {
  name: 'transition',
  props: transitionProps,
  abstract: true,

  render: function render (h) {
    var this$1 = this;

    var children = this.$slots.default;
    if (!children) {
      return
    }

    // filter out text nodes (possible whitespaces)
    children = children.filter(isNotTextNode);
    /* istanbul ignore if */
    if (!children.length) {
      return
    }

    // warn multiple elements
    if (children.length > 1) {
      warn(
        '<transition> can only be used on a single element. Use ' +
        '<transition-group> for lists.',
        this.$parent
      );
    }

    var mode = this.mode;

    // warn invalid mode
    if (mode && mode !== 'in-out' && mode !== 'out-in'
    ) {
      warn(
```

```
    'invalid <transition> mode: ' + mode,
    this.$parent
  );
}

var rawChild = children[0];

// if this is a component root node and the component's
// parent container node also has transition, skip.
if (hasParentTransition(this.$vnode)) {
  return rawChild
}

// apply transition data to child
// use getRealChild() to ignore abstract components e.g. keep-alive
var child = getRealChild(rawChild);
/* istanbul ignore if */
if (!child) {
  return rawChild
}

if (this._leaving) {
  return placeholder(h, rawChild)
}

// ensure a key that is unique to the vnode type and to this transition
// component instance. This key will be used to remove pending leaving nodes
// during entering.
var id = "__transition-" + (this._uid) + "-";
child.key = child.key == null
  ? child.isComment
    ? id + 'comment'
    : id + child.tag
  : isPrimitive(child.key)
    ? (String(child.key).indexOf(id) === 0 ? child.key : id + child.key)
    : child.key;

var data = (child.data || (child.data = {})).transition = extractTransitionData(this);
var oldRawChild = this._vnode;
var oldChild = getRealChild(oldRawChild);

// mark v-show
// so that the transition module can hand over the control to the directive
if (child.data.directives && child.data.directives.some(isVShowDirective)) {
  child.data.show = true;
}

if (
  oldChild &&
  oldChild.data &&
  !isSameChild(child, oldChild) &&
  !isAsyncPlaceholder(oldChild) &&
  // #6687 component root is a comment node
  !(oldChild.componentInstance && oldChild.componentInstance._vnode.isComment)
) {
  // replace old child transition data with fresh one
  // important for dynamic transitions!
  var oldData = oldChild.data.transition = extend({}, data);
  // handle transition mode
  if (mode === 'out-in') {
    // return placeholder node and queue update when leave finishes
    this._leaving = true;
```

```
        mergeVNodeHook(oldData, 'afterLeave', function () {
          this$1._leaving = false;
          this$1.$forceUpdate();
        });
        return placeholder(h, rawChild)
      } else if (mode === 'in-out') {
        if (isAsyncPlaceholder(child)) {
          return oldRawChild
        }
        var delayedLeave;
        var performLeave = function () { delayedLeave(); };
        mergeVNodeHook(data, 'afterEnter', performLeave);
        mergeVNodeHook(data, 'enterCancelled', performLeave);
        mergeVNodeHook(oldData, 'delayLeave', function (leave) { delayedLeave = leave; });
      }
    }

    return rawChild
  }
};

/*  */

var props = extend({
  tag: String,
  moveClass: String
}, transitionProps);

delete props.mode;

var TransitionGroup = {
  props: props,

  beforeMount: function beforeMount () {
    var this$1 = this;

    var update = this._update;
    this._update = function (vnode, hydrating) {
      var restoreActiveInstance = setActiveInstance(this$1);
      // force removing pass
      this$1.__patch__(
        this$1._vnode,
        this$1.kept,
        false, // hydrating
        true // removeOnly (!important, avoids unnecessary moves)
      );
      this$1._vnode = this$1.kept;
      restoreActiveInstance();
      update.call(this$1, vnode, hydrating);
    };
  },

  render: function render (h) {
    var tag = this.tag || this.$vnode.data.tag || 'span';
    var map = Object.create(null);
    var prevChildren = this.prevChildren = this.children;
    var rawChildren = this.$slots.default || [];
    var children = this.children = [];
    var transitionData = extractTransitionData(this);

    for (var i = 0; i < rawChildren.length; i++) {
      var c = rawChildren[i];
```

```
    if (c.tag) {
      if (c.key != null && String(c.key).indexOf('__vlist') !== 0) {
        children.push(c);
        map[c.key] = c
        ;(c.data || (c.data = {})).transition = transitionData;
      } else {
        var opts = c.componentOptions;
        var name = opts ? (opts.Ctor.options.name || opts.tag || '') : c.tag;
        warn(("<transition-group> children must be keyed: <" + name + ">"));
      }
    }
  }

  if (prevChildren) {
    var kept = [];
    var removed = [];
    for (var i$1 = 0; i$1 < prevChildren.length; i$1++) {
      var c$1 = prevChildren[i$1];
      c$1.data.transition = transitionData;
      c$1.data.pos = c$1.elm.getBoundingClientRect();
      if (map[c$1.key]) {
        kept.push(c$1);
      } else {
        removed.push(c$1);
      }
    }
    this.kept = h(tag, null, kept);
    this.removed = removed;
  }

  return h(tag, null, children)
},

updated: function updated () {
  var children = this.prevChildren;
  var moveClass = this.moveClass || ((this.name || 'v') + '-move');
  if (!children.length || !this.hasMove(children[0].elm, moveClass)) {
    return
  }

  // we divide the work into three loops to avoid mixing DOM reads and writes
  // in each iteration - which helps prevent layout thrashing.
  children.forEach(callPendingCbs);
  children.forEach(recordPosition);
  children.forEach(applyTranslation);

  // force reflow to put everything in position
  // assign to this to avoid being removed in tree-shaking
  // $flow-disable-line
  this._reflow = document.body.offsetHeight;

  children.forEach(function (c) {
    if (c.data.moved) {
      var el = c.elm;
      var s = el.style;
      addTransitionClass(el, moveClass);
      s.transform = s.WebkitTransform = s.transitionDuration = '';
      el.addEventListener(transitionEndEvent, el._moveCb = function cb (e) {
        if (e && e.target !== el) {
          return
        }
        if (!e || /transform$/.test(e.propertyName)) {
```

```
            el.removeEventListener(transitionEndEvent, cb);
            el._moveCb = null;
            removeTransitionClass(el, moveClass);
          }
        });
      }
    });
  },

  methods: {
    hasMove: function hasMove (el, moveClass) {
      /* istanbul ignore if */
      if (!hasTransition) {
        return false
      }
      /* istanbul ignore if */
      if (this._hasMove) {
        return this._hasMove
      }
      // Detect whether an element with the move class applied has
      // CSS transitions. Since the element may be inside an entering
      // transition at this very moment, we make a clone of it and remove
      // all other transition classes applied to ensure only the move class
      // is applied.
      var clone = el.cloneNode();
      if (el._transitionClasses) {
        el._transitionClasses.forEach(function (cls) { removeClass(clone, cls); });
      }
      addClass(clone, moveClass);
      clone.style.display = 'none';
      this.$el.appendChild(clone);
      var info = getTransitionInfo(clone);
      this.$el.removeChild(clone);
      return (this._hasMove = info.hasTransform)
    }
  }
};

function callPendingCbs (c) {
  /* istanbul ignore if */
  if (c.elm._moveCb) {
    c.elm._moveCb();
  }
  /* istanbul ignore if */
  if (c.elm._enterCb) {
    c.elm._enterCb();
  }
}

function recordPosition (c) {
  c.data.newPos = c.elm.getBoundingClientRect();
}

function applyTranslation (c) {
  var oldPos = c.data.pos;
  var newPos = c.data.newPos;
  var dx = oldPos.left - newPos.left;
  var dy = oldPos.top - newPos.top;
  if (dx || dy) {
    c.data.moved = true;
    var s = c.elm.style;
    s.transform = s.WebkitTransform = "translate(" + dx + "px," + dy + "px)";
```

```
    s.transitionDuration = '0s';
  }
}

var platformComponents = {
  Transition: Transition,
  TransitionGroup: TransitionGroup
};

/*  */

// install platform specific utils
Vue.config.mustUseProp = mustUseProp;
Vue.config.isReservedTag = isReservedTag;
Vue.config.isReservedAttr = isReservedAttr;
Vue.config.getTagNamespace = getTagNamespace;
Vue.config.isUnknownElement = isUnknownElement;

// install platform runtime directives & components
extend(Vue.options.directives, platformDirectives);
extend(Vue.options.components, platformComponents);

// install platform patch function
Vue.prototype.__patch__ = inBrowser ? patch : noop;

// public mount method
Vue.prototype.$mount = function (
  el,
  hydrating
) {
  el = el && inBrowser ? query(el) : undefined;
  return mountComponent(this, el, hydrating)
};

// devtools global hook
/* istanbul ignore next */
if (inBrowser) {
  setTimeout(function () {
    if (config.devtools) {
      if (devtools) {
        devtools.emit('init', Vue);
      } else {
        console[console.info ? 'info' : 'log'](
          'Download the Vue Devtools extension for a better development experience:\n' +
          'https://github.com/vuejs/vue-devtools'
        );
      }
    }
    if (config.productionTip !== false &&
      typeof console !== 'undefined'
    ) {
      console[console.info ? 'info' : 'log'](
        "You are running Vue in development mode.\n" +
        "Make sure to turn on production mode when deploying for production.\n" +
        "See more tips at https://vuejs.org/guide/deployment.html"
      );
    }
  }, 0);
}

/*  */
```

```
var defaultTagRE = /\{\{((?:.|\r?\n)+?)\}\}/g;
var regexEscapeRE = /[-.*+?^${}()|[\]\/\\]/g;

var buildRegex = cached(function (delimiters) {
  var open = delimiters[0].replace(regexEscapeRE, '\\$&');
  var close = delimiters[1].replace(regexEscapeRE, '\\$&');
  return new RegExp(open + '((?:.|\\n)+?)' + close, 'g')
});



function parseText (
  text,
  delimiters
) {
  var tagRE = delimiters ? buildRegex(delimiters) : defaultTagRE;
  if (!tagRE.test(text)) {
    return
  }
  var tokens = [];
  var rawTokens = [];
  var lastIndex = tagRE.lastIndex = 0;
  var match, index, tokenValue;
  while ((match = tagRE.exec(text))) {
    index = match.index;
    // push text token
    if (index > lastIndex) {
      rawTokens.push(tokenValue = text.slice(lastIndex, index));
      tokens.push(JSON.stringify(tokenValue));
    }
    // tag token
    var exp = parseFilters(match[1].trim());
    tokens.push(("_s(" + exp + ")"));
    rawTokens.push({ '@binding': exp });
    lastIndex = index + match[0].length;
  }
  if (lastIndex < text.length) {
    rawTokens.push(tokenValue = text.slice(lastIndex));
    tokens.push(JSON.stringify(tokenValue));
  }
  return {
    expression: tokens.join('+'),
    tokens: rawTokens
  }
}

/*  */

function transformNode (el, options) {
  var warn = options.warn || baseWarn;
  var staticClass = getAndRemoveAttr(el, 'class');
  if (staticClass) {
    var res = parseText(staticClass, options.delimiters);
    if (res) {
      warn(
        "class=\"" + staticClass + "\": " +
        'Interpolation inside attributes has been removed. ' +
        'Use v-bind or the colon shorthand instead. For example, ' +
        'instead of <div class="{{ val }}">, use <div :class="val">.',
        el.rawAttrsMap['class']
      );
    }
```

```
  }
  if (staticClass) {
    el.staticClass = JSON.stringify(staticClass);
  }
  var classBinding = getBindingAttr(el, 'class', false /* getStatic */);
  if (classBinding) {
    el.classBinding = classBinding;
  }
}

function genData (el) {
  var data = '';
  if (el.staticClass) {
    data += "staticClass:" + (el.staticClass) + ",";
  }
  if (el.classBinding) {
    data += "class:" + (el.classBinding) + ",";
  }
  return data
}

var klass$1 = {
  staticKeys: ['staticClass'],
  transformNode: transformNode,
  genData: genData
};

/*  */

function transformNode$1 (el, options) {
  var warn = options.warn || baseWarn;
  var staticStyle = getAndRemoveAttr(el, 'style');
  if (staticStyle) {
    /* istanbul ignore if */
    {
      var res = parseText(staticStyle, options.delimiters);
      if (res) {
        warn(
          "style=\"" + staticStyle + "\": " +
          'Interpolation inside attributes has been removed. ' +
          'Use v-bind or the colon shorthand instead. For example, ' +
          'instead of <div style="{{ val }}">, use <div :style="val">.',
          el.rawAttrsMap['style']
        );
      }
    }
    el.staticStyle = JSON.stringify(parseStyleText(staticStyle));
  }

  var styleBinding = getBindingAttr(el, 'style', false /* getStatic */);
  if (styleBinding) {
    el.styleBinding = styleBinding;
  }
}

function genData$1 (el) {
  var data = '';
  if (el.staticStyle) {
    data += "staticStyle:" + (el.staticStyle) + ",";
  }
  if (el.styleBinding) {
    data += "style:(" + (el.styleBinding) + "),";
```

```
  }
  return data
}

var style$1 = {
  staticKeys: ['staticStyle'],
  transformNode: transformNode$1,
  genData: genData$1
};

/*  */

var decoder;

var he = {
  decode: function decode (html) {
    decoder = decoder || document.createElement('div');
    decoder.innerHTML = html;
    return decoder.textContent
  }
};

/*  */

var isUnaryTag = makeMap(
  'area,base,br,col,embed,frame,hr,img,input,isindex,keygen,' +
  'link,meta,param,source,track,wbr'
);

// Elements that you can, intentionally, leave open
// (and which close themselves)
var canBeLeftOpenTag = makeMap(
  'colgroup,dd,dt,li,options,p,td,tfoot,th,thead,tr,source'
);

// HTML5 tags https://html.spec.whatwg.org/multipage/indices.html#elements-3
// Phrasing Content https://html.spec.whatwg.org/multipage/dom.html#phrasing-content
var isNonPhrasingTag = makeMap(
  'address,article,aside,base,blockquote,body,caption,col,colgroup,dd,' +
  'details,dialog,div,dl,dt,fieldset,figcaption,figure,footer,form,' +
  'h1,h2,h3,h4,h5,h6,head,header,hgroup,hr,html,legend,li,menuitem,meta,' +
  'optgroup,option,param,rp,rt,source,style,summary,tbody,td,tfoot,th,thead,' +
  'title,tr,track'
);

/**
 * Not type-checking this file because it's mostly vendor code.
 */

// Regular Expressions for parsing tags and attributes
var attribute = /^\s*([^\s"'<>\/=]+)(?:\s*(=)\s*(?:"([^"]*)"+|'([^']*)'+|([^\s"'=<>`]+)))?/;
var dynamicArgAttribute = /^\s*((?:v-[\w-]+:|@|:|#)\[[^=]+\][^\s"'<>\/=]*)(?:\s*(=)\s*(?:"([^
var ncname = "[a-zA-Z_][\\-\\.0-9_a-zA-Z" + (unicodeRegExp.source) + "]*";
var qnameCapture = "((?:" + ncname + "\\:)?" + ncname + ")";
var startTagOpen = new RegExp(("^<" + qnameCapture));
var startTagClose = /^\s*(\/?)>/;
var endTag = new RegExp(("^<\\/" + qnameCapture + "[^>]*>"));
var doctype = /^<!DOCTYPE [^>]+>/i;
// #7298: escape - to avoid being pased as HTML comment when inlined in page
var comment = /^<!\--/;
var conditionalComment = /^<!\[/;
```

```
// Special Elements (can contain anything)
var isPlainTextElement = makeMap('script,style,textarea', true);
var reCache = {};

var decodingMap = {
  '&lt;': '<',
  '&gt;': '>',
  '&quot;': '"',
  '&amp;': '&',
  '&#10;': '\n',
  '&#9;': '\t',
  '&#39;': "'"
};
var encodedAttr = /&(?:lt|gt|quot|amp|#39);/g;
var encodedAttrWithNewLines = /&(?:lt|gt|quot|amp|#39|#10|#9);/g;

// #5992
var isIgnoreNewlineTag = makeMap('pre,textarea', true);
var shouldIgnoreFirstNewline = function (tag, html) { return tag && isIgnoreNewlineTag(tag) &

function decodeAttr (value, shouldDecodeNewlines) {
  var re = shouldDecodeNewlines ? encodedAttrWithNewLines : encodedAttr;
  return value.replace(re, function (match) { return decodingMap[match]; })
}

function parseHTML (html, options) {
  var stack = [];
  var expectHTML = options.expectHTML;
  var isUnaryTag$$1 = options.isUnaryTag || no;
  var canBeLeftOpenTag$$1 = options.canBeLeftOpenTag || no;
  var index = 0;
  var last, lastTag;
  while (html) {
    last = html;
    // Make sure we're not in a plaintext content element like script/style
    if (!lastTag || !isPlainTextElement(lastTag)) {
      var textEnd = html.indexOf('<');
      if (textEnd === 0) {
        // Comment:
        if (comment.test(html)) {
          var commentEnd = html.indexOf('-->');

          if (commentEnd >= 0) {
            if (options.shouldKeepComment) {
              options.comment(html.substring(4, commentEnd), index, index + commentEnd + 3);
            }
            advance(commentEnd + 3);
            continue
          }
        }

        // http://en.wikipedia.org/wiki/Conditional_comment#Downlevel-revealed_conditional_co
        if (conditionalComment.test(html)) {
          var conditionalEnd = html.indexOf(']>');

          if (conditionalEnd >= 0) {
            advance(conditionalEnd + 2);
            continue
          }
        }

        // Doctype:
```

```
      var doctypeMatch = html.match(doctype);
      if (doctypeMatch) {
        advance(doctypeMatch[0].length);
        continue
      }

      // End tag:
      var endTagMatch = html.match(endTag);
      if (endTagMatch) {
        var curIndex = index;
        advance(endTagMatch[0].length);
        parseEndTag(endTagMatch[1], curIndex, index);
        continue
      }

      // Start tag:
      var startTagMatch = parseStartTag();
      if (startTagMatch) {
        handleStartTag(startTagMatch);
        if (shouldIgnoreFirstNewline(startTagMatch.tagName, html)) {
          advance(1);
        }
        continue
      }
    }

    var text = (void 0), rest = (void 0), next = (void 0);
    if (textEnd >= 0) {
      rest = html.slice(textEnd);
      while (
        !endTag.test(rest) &&
        !startTagOpen.test(rest) &&
        !comment.test(rest) &&
        !conditionalComment.test(rest)
      ) {
        // < in plain text, be forgiving and treat it as text
        next = rest.indexOf('<', 1);
        if (next < 0) { break }
        textEnd += next;
        rest = html.slice(textEnd);
      }
      text = html.substring(0, textEnd);
    }

    if (textEnd < 0) {
      text = html;
    }

    if (text) {
      advance(text.length);
    }

    if (options.chars && text) {
      options.chars(text, index - text.length, index);
    }
  } else {
    var endTagLength = 0;
    var stackedTag = lastTag.toLowerCase();
    var reStackedTag = reCache[stackedTag] || (reCache[stackedTag] = new RegExp('([\\s\\S]*
    var rest$1 = html.replace(reStackedTag, function (all, text, endTag) {
      endTagLength = endTag.length;
      if (!isPlainTextElement(stackedTag) && stackedTag !== 'noscript') {
```

```
        text = text
          .replace(/<!\--([\s\S]*?)-->/g, '$1') // #7298
          .replace(/<!\[CDATA\[([\s\S]*?)]]>/g, '$1');
      }
      if (shouldIgnoreFirstNewline(stackedTag, text)) {
        text = text.slice(1);
      }
      if (options.chars) {
        options.chars(text);
      }
      return ''
    });
    index += html.length - rest$1.length;
    html = rest$1;
    parseEndTag(stackedTag, index - endTagLength, index);
  }

  if (html === last) {
    options.chars && options.chars(html);
    if (!stack.length && options.warn) {
      options.warn(("Mal-formatted tag at end of template: \"" + html + "\""), { start: ind
    }
    break
  }
}

// Clean up any remaining tags
parseEndTag();

function advance (n) {
  index += n;
  html = html.substring(n);
}

function parseStartTag () {
  var start = html.match(startTagOpen);
  if (start) {
    var match = {
      tagName: start[1],
      attrs: [],
      start: index
    };
    advance(start[0].length);
    var end, attr;
    while (!(end = html.match(startTagClose)) && (attr = html.match(dynamicArgAttribute) ||
      attr.start = index;
      advance(attr[0].length);
      attr.end = index;
      match.attrs.push(attr);
    }
    if (end) {
      match.unarySlash = end[1];
      advance(end[0].length);
      match.end = index;
      return match
    }
  }
}

function handleStartTag (match) {
  var tagName = match.tagName;
  var unarySlash = match.unarySlash;
```

```
  if (expectHTML) {
    if (lastTag === 'p' && isNonPhrasingTag(tagName)) {
      parseEndTag(lastTag);
    }
    if (canBeLeftOpenTag$$1(tagName) && lastTag === tagName) {
      parseEndTag(tagName);
    }
  }

  var unary = isUnaryTag$$1(tagName) || !!unarySlash;

  var l = match.attrs.length;
  var attrs = new Array(l);
  for (var i = 0; i < l; i++) {
    var args = match.attrs[i];
    var value = args[3] || args[4] || args[5] || '';
    var shouldDecodeNewlines = tagName === 'a' && args[1] === 'href'
      ? options.shouldDecodeNewlinesForHref
      : options.shouldDecodeNewlines;
    attrs[i] = {
      name: args[1],
      value: decodeAttr(value, shouldDecodeNewlines)
    };
    if (options.outputSourceRange) {
      attrs[i].start = args.start + args[0].match(/^\s*/).length;
      attrs[i].end = args.end;
    }
  }

  if (!unary) {
    stack.push({ tag: tagName, lowerCasedTag: tagName.toLowerCase(), attrs: attrs, start: m
    lastTag = tagName;
  }

  if (options.start) {
    options.start(tagName, attrs, unary, match.start, match.end);
  }
}

function parseEndTag (tagName, start, end) {
  var pos, lowerCasedTagName;
  if (start == null) { start = index; }
  if (end == null) { end = index; }

  // Find the closest opened tag of the same type
  if (tagName) {
    lowerCasedTagName = tagName.toLowerCase();
    for (pos = stack.length - 1; pos >= 0; pos--) {
      if (stack[pos].lowerCasedTag === lowerCasedTagName) {
        break
      }
    }
  } else {
    // If no tag name is provided, clean shop
    pos = 0;
  }

  if (pos >= 0) {
    // Close all the open elements, up the stack
    for (var i = stack.length - 1; i >= pos; i--) {
      if (i > pos || !tagName &&
```

```
          options.warn
        ) {
          options.warn(
            ("tag <" + (stack[i].tag) + "> has no matching end tag."),
            { start: stack[i].start, end: stack[i].end }
          );
        }
        if (options.end) {
          options.end(stack[i].tag, start, end);
        }
      }

      // Remove the open elements from the stack
      stack.length = pos;
      lastTag = pos && stack[pos - 1].tag;
    } else if (lowerCasedTagName === 'br') {
      if (options.start) {
        options.start(tagName, [], true, start, end);
      }
    } else if (lowerCasedTagName === 'p') {
      if (options.start) {
        options.start(tagName, [], false, start, end);
      }
      if (options.end) {
        options.end(tagName, start, end);
      }
    }
  }
}

/*  */

var onRE = /^@|^v-on:/;
var dirRE = /^v-|^@|^:/;
var forAliasRE = /([\s\S]*?)\s+(?:in|of)\s+([\s\S]*)/;
var forIteratorRE = /,([^,\}\]]*)(?:,([^,\}\]]*))?$/;
var stripParensRE = /^\(|\)$/g;
var dynamicArgRE = /^\[.*\]$/;

var argRE = /:(.*)$/;
var bindRE = /^:|^\.|^v-bind:/;
var modifierRE = /\.[^.\]]+(?=[^\]]*$)/g;

var slotRE = /^v-slot(:|$)|^#/;

var lineBreakRE = /[\r\n]/;
var whitespaceRE$1 = /\s+/g;

var invalidAttributeRE = /[\s"'<>\/=]/;

var decodeHTMLCached = cached(he.decode);

var emptySlotScopeToken = "_empty_";

// configurable state
var warn$2;
var delimiters;
var transforms;
var preTransforms;
var postTransforms;
var platformIsPreTag;
var platformMustUseProp;
```

```
var platformGetTagNamespace;
var maybeComponent;

function createASTElement (
  tag,
  attrs,
  parent
) {
  return {
    type: 1,
    tag: tag,
    attrsList: attrs,
    attrsMap: makeAttrsMap(attrs),
    rawAttrsMap: {},
    parent: parent,
    children: []
  }
}

/**
 * Convert HTML string to AST.
 */
function parse (
  template,
  options
) {
  warn$2 = options.warn || baseWarn;

  platformIsPreTag = options.isPreTag || no;
  platformMustUseProp = options.mustUseProp || no;
  platformGetTagNamespace = options.getTagNamespace || no;
  var isReservedTag = options.isReservedTag || no;
  maybeComponent = function (el) { return !!el.component || !isReservedTag(el.tag); };

  transforms = pluckModuleFunction(options.modules, 'transformNode');
  preTransforms = pluckModuleFunction(options.modules, 'preTransformNode');
  postTransforms = pluckModuleFunction(options.modules, 'postTransformNode');

  delimiters = options.delimiters;

  var stack = [];
  var preserveWhitespace = options.preserveWhitespace !== false;
  var whitespaceOption = options.whitespace;
  var root;
  var currentParent;
  var inVPre = false;
  var inPre = false;
  var warned = false;

  function warnOnce (msg, range) {
    if (!warned) {
      warned = true;
      warn$2(msg, range);
    }
  }

  function closeElement (element) {
    trimEndingWhitespace(element);
    if (!inVPre && !element.processed) {
      element = processElement(element, options);
    }
    // tree management
```

```
  if (!stack.length && element !== root) {
    // allow root elements with v-if, v-else-if and v-else
    if (root.if && (element.elseif || element.else)) {
      {
        checkRootConstraints(element);
      }
      addIfCondition(root, {
        exp: element.elseif,
        block: element
      });
    } else {
      warnOnce(
        "Component template should contain exactly one root element. " +
        "If you are using v-if on multiple elements, " +
        "use v-else-if to chain them instead.",
        { start: element.start }
      );
    }
  }
  if (currentParent && !element.forbidden) {
    if (element.elseif || element.else) {
      processIfConditions(element, currentParent);
    } else {
      if (element.slotScope) {
        // scoped slot
        // keep it in the children list so that v-else(-if) conditions can
        // find it as the prev node.
        var name = element.slotTarget || '"default"'
        ;(currentParent.scopedSlots || (currentParent.scopedSlots = {}))[name] = element;
      }
      currentParent.children.push(element);
      element.parent = currentParent;
    }
  }

  // final children cleanup
  // filter out scoped slots
  element.children = element.children.filter(function (c) { return !(c).slotScope; });
  // remove trailing whitespace node again
  trimEndingWhitespace(element);

  // check pre state
  if (element.pre) {
    inVPre = false;
  }
  if (platformIsPreTag(element.tag)) {
    inPre = false;
  }
  // apply post-transforms
  for (var i = 0; i < postTransforms.length; i++) {
    postTransforms[i](element, options);
  }
}

function trimEndingWhitespace (el) {
  // remove trailing whitespace node
  if (!inPre) {
    var lastNode;
    while (
      (lastNode = el.children[el.children.length - 1]) &&
      lastNode.type === 3 &&
      lastNode.text === ' '
```

```
    ) {
      el.children.pop();
    }
  }
}

function checkRootConstraints (el) {
  if (el.tag === 'slot' || el.tag === 'template') {
    warnOnce(
      "Cannot use <" + (el.tag) + "> as component root element because it may " +
      'contain multiple nodes.',
      { start: el.start }
    );
  }
  if (el.attrsMap.hasOwnProperty('v-for')) {
    warnOnce(
      'Cannot use v-for on stateful component root element because ' +
      'it renders multiple elements.',
      el.rawAttrsMap['v-for']
    );
  }
}

parseHTML(template, {
  warn: warn$2,
  expectHTML: options.expectHTML,
  isUnaryTag: options.isUnaryTag,
  canBeLeftOpenTag: options.canBeLeftOpenTag,
  shouldDecodeNewlines: options.shouldDecodeNewlines,
  shouldDecodeNewlinesForHref: options.shouldDecodeNewlinesForHref,
  shouldKeepComment: options.comments,
  outputSourceRange: options.outputSourceRange,
  start: function start (tag, attrs, unary, start$1, end) {
    // check namespace.
    // inherit parent ns if there is one
    var ns = (currentParent && currentParent.ns) || platformGetTagNamespace(tag);

    // handle IE svg bug
    /* istanbul ignore if */
    if (isIE && ns === 'svg') {
      attrs = guardIESVGBug(attrs);
    }

    var element = createASTElement(tag, attrs, currentParent);
    if (ns) {
      element.ns = ns;
    }

    {
      if (options.outputSourceRange) {
        element.start = start$1;
        element.end = end;
        element.rawAttrsMap = element.attrsList.reduce(function (cumulated, attr) {
          cumulated[attr.name] = attr;
          return cumulated
        }, {});
      }
      attrs.forEach(function (attr) {
        if (invalidAttributeRE.test(attr.name)) {
          warn$2(
            "Invalid dynamic argument expression: attribute names cannot contain " +
            "spaces, quotes, <, >, / or =.",
```

```
            {
              start: attr.start + attr.name.indexOf("["),
              end: attr.start + attr.name.length
            }
          );
        }
      });
    }

    if (isForbiddenTag(element) && !isServerRendering()) {
      element.forbidden = true;
      warn$2(
        'Templates should only be responsible for mapping the state to the ' +
        'UI. Avoid placing tags with side-effects in your templates, such as ' +
        "<" + tag + ">" + ', as they will not be parsed.',
        { start: element.start }
      );
    }

    // apply pre-transforms
    for (var i = 0; i < preTransforms.length; i++) {
      element = preTransforms[i](element, options) || element;
    }

    if (!inVPre) {
      processPre(element);
      if (element.pre) {
        inVPre = true;
      }
    }
    if (platformIsPreTag(element.tag)) {
      inPre = true;
    }
    if (inVPre) {
      processRawAttrs(element);
    } else if (!element.processed) {
      // structural directives
      processFor(element);
      processIf(element);
      processOnce(element);
    }

    if (!root) {
      root = element;
      {
        checkRootConstraints(root);
      }
    }

    if (!unary) {
      currentParent = element;
      stack.push(element);
    } else {
      closeElement(element);
    }
  },

  end: function end (tag, start, end$1) {
    var element = stack[stack.length - 1];
    // pop stack
    stack.length -= 1;
    currentParent = stack[stack.length - 1];
```

```
    if (options.outputSourceRange) {
      element.end = end$1;
    }
    closeElement(element);
  },

  chars: function chars (text, start, end) {
    if (!currentParent) {
      {
        if (text === template) {
          warnOnce(
            'Component template requires a root element, rather than just text.',
            { start: start }
          );
        } else if ((text = text.trim())) {
          warnOnce(
            ("text \"" + text + "\" outside root element will be ignored."),
            { start: start }
          );
        }
      }
      return
    }
    // IE textarea placeholder bug
    /* istanbul ignore if */
    if (isIE &&
      currentParent.tag === 'textarea' &&
      currentParent.attrsMap.placeholder === text
    ) {
      return
    }
    var children = currentParent.children;
    if (inPre || text.trim()) {
      text = isTextTag(currentParent) ? text : decodeHTMLCached(text);
    } else if (!children.length) {
      // remove the whitespace-only node right after an opening tag
      text = '';
    } else if (whitespaceOption) {
      if (whitespaceOption === 'condense') {
        // in condense mode, remove the whitespace node if it contains
        // line break, otherwise condense to a single space
        text = lineBreakRE.test(text) ? '' : ' ';
      } else {
        text = ' ';
      }
    } else {
      text = preserveWhitespace ? ' ' : '';
    }
    if (text) {
      if (!inPre && whitespaceOption === 'condense') {
        // condense consecutive whitespaces into single space
        text = text.replace(whitespaceRE$1, ' ');
      }
      var res;
      var child;
      if (!inVPre && text !== ' ' && (res = parseText(text, delimiters))) {
        child = {
          type: 2,
          expression: res.expression,
          tokens: res.tokens,
          text: text
        };
```

```
        } else if (text !== ' ' || !children.length || children[children.length - 1].text !==
          child = {
            type: 3,
            text: text
          };
        }
        if (child) {
          if (options.outputSourceRange) {
            child.start = start;
            child.end = end;
          }
          children.push(child);
        }
      }
    },
    comment: function comment (text, start, end) {
      // adding anyting as a sibling to the root node is forbidden
      // comments should still be allowed, but ignored
      if (currentParent) {
        var child = {
          type: 3,
          text: text,
          isComment: true
        };
        if (options.outputSourceRange) {
          child.start = start;
          child.end = end;
        }
        currentParent.children.push(child);
      }
    }
  });
  return root
}

function processPre (el) {
  if (getAndRemoveAttr(el, 'v-pre') != null) {
    el.pre = true;
  }
}

function processRawAttrs (el) {
  var list = el.attrsList;
  var len = list.length;
  if (len) {
    var attrs = el.attrs = new Array(len);
    for (var i = 0; i < len; i++) {
      attrs[i] = {
        name: list[i].name,
        value: JSON.stringify(list[i].value)
      };
      if (list[i].start != null) {
        attrs[i].start = list[i].start;
        attrs[i].end = list[i].end;
      }
    }
  } else if (!el.pre) {
    // non root node in pre blocks with no attributes
    el.plain = true;
  }
}
```

```
function processElement (
  element,
  options
) {
  processKey(element);

  // determine whether this is a plain element after
  // removing structural attributes
  element.plain = (
    !element.key &&
    !element.scopedSlots &&
    !element.attrsList.length
  );

  processRef(element);
  processSlotContent(element);
  processSlotOutlet(element);
  processComponent(element);
  for (var i = 0; i < transforms.length; i++) {
    element = transforms[i](element, options) || element;
  }
  processAttrs(element);
  return element
}

function processKey (el) {
  var exp = getBindingAttr(el, 'key');
  if (exp) {
    {
      if (el.tag === 'template') {
        warn$2(
          "<template> cannot be keyed. Place the key on real elements instead.",
          getRawBindingAttr(el, 'key')
        );
      }
      if (el.for) {
        var iterator = el.iterator2 || el.iterator1;
        var parent = el.parent;
        if (iterator && iterator === exp && parent && parent.tag === 'transition-group') {
          warn$2(
            "Do not use v-for index as key on <transition-group> children, " +
            "this is the same as not using keys.",
            getRawBindingAttr(el, 'key'),
            true /* tip */
          );
        }
      }
    }
    el.key = exp;
  }
}

function processRef (el) {
  var ref = getBindingAttr(el, 'ref');
  if (ref) {
    el.ref = ref;
    el.refInFor = checkInFor(el);
  }
}

function processFor (el) {
  var exp;
```

```
  if ((exp = getAndRemoveAttr(el, 'v-for'))) {
    var res = parseFor(exp);
    if (res) {
      extend(el, res);
    } else {
      warn$2(
        ("Invalid v-for expression: " + exp),
        el.rawAttrsMap['v-for']
      );
    }
  }
}



function parseFor (exp) {
  var inMatch = exp.match(forAliasRE);
  if (!inMatch) { return }
  var res = {};
  res.for = inMatch[2].trim();
  var alias = inMatch[1].trim().replace(stripParensRE, '');
  var iteratorMatch = alias.match(forIteratorRE);
  if (iteratorMatch) {
    res.alias = alias.replace(forIteratorRE, '').trim();
    res.iterator1 = iteratorMatch[1].trim();
    if (iteratorMatch[2]) {
      res.iterator2 = iteratorMatch[2].trim();
    }
  } else {
    res.alias = alias;
  }
  return res
}

function processIf (el) {
  var exp = getAndRemoveAttr(el, 'v-if');
  if (exp) {
    el.if = exp;
    addIfCondition(el, {
      exp: exp,
      block: el
    });
  } else {
    if (getAndRemoveAttr(el, 'v-else') != null) {
      el.else = true;
    }
    var elseif = getAndRemoveAttr(el, 'v-else-if');
    if (elseif) {
      el.elseif = elseif;
    }
  }
}

function processIfConditions (el, parent) {
  var prev = findPrevElement(parent.children);
  if (prev && prev.if) {
    addIfCondition(prev, {
      exp: el.elseif,
      block: el
    });
  } else {
    warn$2(
```

```
      "v-" + (el.elseif ? ('else-if="' + el.elseif + '"') : 'else') + " " +
      "used on element <" + (el.tag) + "> without corresponding v-if.",
      el.rawAttrsMap[el.elseif ? 'v-else-if' : 'v-else']
    );
  }
}

function findPrevElement (children) {
  var i = children.length;
  while (i--) {
    if (children[i].type === 1) {
      return children[i]
    } else {
      if (children[i].text !== ' ') {
        warn$2(
          "text \"" + (children[i].text.trim()) + "\" between v-if and v-else(-if) " +
          "will be ignored.",
          children[i]
        );
      }
      children.pop();
    }
  }
}

function addIfCondition (el, condition) {
  if (!el.ifConditions) {
    el.ifConditions = [];
  }
  el.ifConditions.push(condition);
}

function processOnce (el) {
  var once$$1 = getAndRemoveAttr(el, 'v-once');
  if (once$$1 != null) {
    el.once = true;
  }
}

// handle content being passed to a component as slot,
// e.g. <template slot="xxx">, <div slot-scope="xxx">
function processSlotContent (el) {
  var slotScope;
  if (el.tag === 'template') {
    slotScope = getAndRemoveAttr(el, 'scope');
    /* istanbul ignore if */
    if (slotScope) {
      warn$2(
        "the \"scope\" attribute for scoped slots have been deprecated and " +
        "replaced by \"slot-scope\" since 2.5. The new \"slot-scope\" attribute " +
        "can also be used on plain elements in addition to <template> to " +
        "denote scoped slots.",
        el.rawAttrsMap['scope'],
        true
      );
    }
    el.slotScope = slotScope || getAndRemoveAttr(el, 'slot-scope');
  } else if ((slotScope = getAndRemoveAttr(el, 'slot-scope'))) {
    /* istanbul ignore if */
    if (el.attrsMap['v-for']) {
      warn$2(
        "Ambiguous combined usage of slot-scope and v-for on <" + (el.tag) + "> " +
```

```
      "(v-for takes higher priority). Use a wrapper <template> for the " +
      "scoped slot to make it clearer.",
      el.rawAttrsMap['slot-scope'],
      true
    );
  }
  el.slotScope = slotScope;
}

// slot="xxx"
var slotTarget = getBindingAttr(el, 'slot');
if (slotTarget) {
  el.slotTarget = slotTarget === '""' ? '"default"' : slotTarget;
  el.slotTargetDynamic = !!(el.attrsMap[':slot'] || el.attrsMap['v-bind:slot']);
  // preserve slot as an attribute for native shadow DOM compat
  // only for non-scoped slots.
  if (el.tag !== 'template' && !el.slotScope) {
    addAttr(el, 'slot', slotTarget, getRawBindingAttr(el, 'slot'));
  }
}

// 2.6 v-slot syntax
{
  if (el.tag === 'template') {
    // v-slot on <template>
    var slotBinding = getAndRemoveAttrByRegex(el, slotRE);
    if (slotBinding) {
      {
        if (el.slotTarget || el.slotScope) {
          warn$2(
            "Unexpected mixed usage of different slot syntaxes.",
            el
          );
        }
        if (el.parent && !maybeComponent(el.parent)) {
          warn$2(
            "<template v-slot> can only appear at the root level inside " +
            "the receiving the component",
            el
          );
        }
      }
      var ref = getSlotName(slotBinding);
      var name = ref.name;
      var dynamic = ref.dynamic;
      el.slotTarget = name;
      el.slotTargetDynamic = dynamic;
      el.slotScope = slotBinding.value || emptySlotScopeToken; // force it into a scoped sl
    }
  } else {
    // v-slot on component, denotes default slot
    var slotBinding$1 = getAndRemoveAttrByRegex(el, slotRE);
    if (slotBinding$1) {
      {
        if (!maybeComponent(el)) {
          warn$2(
            "v-slot can only be used on components or <template>.",
            slotBinding$1
          );
        }
        if (el.slotScope || el.slotTarget) {
          warn$2(
```

```
              "Unexpected mixed usage of different slot syntaxes.",
              el
            );
          }
          if (el.scopedSlots) {
            warn$2(
              "To avoid scope ambiguity, the default slot should also use " +
              "<template> syntax when there are other named slots.",
              slotBinding$1
            );
          }
        }
        // add the component's children to its default slot
        var slots = el.scopedSlots || (el.scopedSlots = {});
        var ref$1 = getSlotName(slotBinding$1);
        var name$1 = ref$1.name;
        var dynamic$1 = ref$1.dynamic;
        var slotContainer = slots[name$1] = createASTElement('template', [], el);
        slotContainer.slotTarget = name$1;
        slotContainer.slotTargetDynamic = dynamic$1;
        slotContainer.children = el.children.filter(function (c) {
          if (!c.slotScope) {
            c.parent = slotContainer;
            return true
          }
        });
        slotContainer.slotScope = slotBinding$1.value || emptySlotScopeToken;
        // remove children as they are returned from scopedSlots now
        el.children = [];
        // mark el non-plain so data gets generated
        el.plain = false;
      }
    }
  }
}

function getSlotName (binding) {
  var name = binding.name.replace(slotRE, '');
  if (!name) {
    if (binding.name[0] !== '#') {
      name = 'default';
    } else {
      warn$2(
        "v-slot shorthand syntax requires a slot name.",
        binding
      );
    }
  }
  return dynamicArgRE.test(name)
    // dynamic [name]
    ? { name: name.slice(1, -1), dynamic: true }
    // static name
    : { name: ("\"" + name + "\""), dynamic: false }
}

// handle <slot/> outlets
function processSlotOutlet (el) {
  if (el.tag === 'slot') {
    el.slotName = getBindingAttr(el, 'name');
    if (el.key) {
      warn$2(
        "`key` does not work on <slot> because slots are abstract outlets " +
```

```
        "and can possibly expand into multiple elements. " +
        "Use the key on a wrapping element instead.",
        getRawBindingAttr(el, 'key')
      );
    }
  }
}

function processComponent (el) {
  var binding;
  if ((binding = getBindingAttr(el, 'is'))) {
    el.component = binding;
  }
  if (getAndRemoveAttr(el, 'inline-template') != null) {
    el.inlineTemplate = true;
  }
}

function processAttrs (el) {
  var list = el.attrsList;
  var i, l, name, rawName, value, modifiers, syncGen, isDynamic;
  for (i = 0, l = list.length; i < l; i++) {
    name = rawName = list[i].name;
    value = list[i].value;
    if (dirRE.test(name)) {
      // mark element as dynamic
      el.hasBindings = true;
      // modifiers
      modifiers = parseModifiers(name.replace(dirRE, ''));
      // support .foo shorthand syntax for the .prop modifier
      if (modifiers) {
        name = name.replace(modifierRE, '');
      }
      if (bindRE.test(name)) { // v-bind
        name = name.replace(bindRE, '');
        value = parseFilters(value);
        isDynamic = dynamicArgRE.test(name);
        if (isDynamic) {
          name = name.slice(1, -1);
        }
        if (
          value.trim().length === 0
        ) {
          warn$2(
            ("The value for a v-bind expression cannot be empty. Found in \"v-bind:" + name +
          );
        }
        if (modifiers) {
          if (modifiers.prop && !isDynamic) {
            name = camelize(name);
            if (name === 'innerHtml') { name = 'innerHTML'; }
          }
          if (modifiers.camel && !isDynamic) {
            name = camelize(name);
          }
          if (modifiers.sync) {
            syncGen = genAssignmentCode(value, "$event");
            if (!isDynamic) {
              addHandler(
                el,
                ("update:" + (camelize(name))),
                syncGen,
```

```
            null,
            false,
            warn$2,
            list[i]
          );
          if (hyphenate(name) !== camelize(name)) {
            addHandler(
              el,
              ("update:" + (hyphenate(name))),
              syncGen,
              null,
              false,
              warn$2,
              list[i]
            );
          }
        } else {
          // handler w/ dynamic event name
          addHandler(
            el,
            ("\"update:\"+(" + name + ")"),
            syncGen,
            null,
            false,
            warn$2,
            list[i],
            true // dynamic
          );
        }
      }
    }
    if ((modifiers && modifiers.prop) || (
      !el.component && platformMustUseProp(el.tag, el.attrsMap.type, name)
    )) {
      addProp(el, name, value, list[i], isDynamic);
    } else {
      addAttr(el, name, value, list[i], isDynamic);
    }
  } else if (onRE.test(name)) { // v-on
    name = name.replace(onRE, '');
    isDynamic = dynamicArgRE.test(name);
    if (isDynamic) {
      name = name.slice(1, -1);
    }
    addHandler(el, name, value, modifiers, false, warn$2, list[i], isDynamic);
  } else { // normal directives
    name = name.replace(dirRE, '');
    // parse arg
    var argMatch = name.match(argRE);
    var arg = argMatch && argMatch[1];
    isDynamic = false;
    if (arg) {
      name = name.slice(0, -(arg.length + 1));
      if (dynamicArgRE.test(arg)) {
        arg = arg.slice(1, -1);
        isDynamic = true;
      }
    }
    addDirective(el, name, rawName, value, arg, isDynamic, modifiers, list[i]);
    if (name === 'model') {
      checkForAliasModel(el, value);
    }
```

```
      }
    } else {
      // literal attribute
      {
        var res = parseText(value, delimiters);
        if (res) {
          warn$2(
            name + "=\"" + value + "\": " +
            'Interpolation inside attributes has been removed. ' +
            'Use v-bind or the colon shorthand instead. For example, ' +
            'instead of <div id="{{ val }}">, use <div :id="val">.',
            list[i]
          );
        }
      }
      addAttr(el, name, JSON.stringify(value), list[i]);
      // #6887 firefox doesn't update muted state if set via attribute
      // even immediately after element creation
      if (!el.component &&
          name === 'muted' &&
          platformMustUseProp(el.tag, el.attrsMap.type, name)) {
        addProp(el, name, 'true', list[i]);
      }
    }
  }
}

function checkInFor (el) {
  var parent = el;
  while (parent) {
    if (parent.for !== undefined) {
      return true
    }
    parent = parent.parent;
  }
  return false
}

function parseModifiers (name) {
  var match = name.match(modifierRE);
  if (match) {
    var ret = {};
    match.forEach(function (m) { ret[m.slice(1)] = true; });
    return ret
  }
}

function makeAttrsMap (attrs) {
  var map = {};
  for (var i = 0, l = attrs.length; i < l; i++) {
    if (
      map[attrs[i].name] && !isIE && !isEdge
    ) {
      warn$2('duplicate attribute: ' + attrs[i].name, attrs[i]);
    }
    map[attrs[i].name] = attrs[i].value;
  }
  return map
}

// for script (e.g. type="x/template") or style, do not decode content
function isTextTag (el) {
```

```
  return el.tag === 'script' || el.tag === 'style'
}

function isForbiddenTag (el) {
  return (
    el.tag === 'style' ||
    (el.tag === 'script' && (
      !el.attrsMap.type ||
      el.attrsMap.type === 'text/javascript'
    ))
  )
}

var ieNSBug = /^xmlns:NS\d+/;
var ieNSPrefix = /^NS\d+:/;

/* istanbul ignore next */
function guardIESVGBug (attrs) {
  var res = [];
  for (var i = 0; i < attrs.length; i++) {
    var attr = attrs[i];
    if (!ieNSBug.test(attr.name)) {
      attr.name = attr.name.replace(ieNSPrefix, '');
      res.push(attr);
    }
  }
  return res
}

function checkForAliasModel (el, value) {
  var _el = el;
  while (_el) {
    if (_el.for && _el.alias === value) {
      warn$2(
        "<" + (el.tag) + " v-model=\"" + value + "\">: " +
        "You are binding v-model directly to a v-for iteration alias. " +
        "This will not be able to modify the v-for source array because " +
        "writing to the alias is like modifying a function local variable. " +
        "Consider using an array of objects and use v-model on an object property instead.",
        el.rawAttrsMap['v-model']
      );
    }
    _el = _el.parent;
  }
}

/*  */

function preTransformNode (el, options) {
  if (el.tag === 'input') {
    var map = el.attrsMap;
    if (!map['v-model']) {
      return
    }

    var typeBinding;
    if (map[':type'] || map['v-bind:type']) {
      typeBinding = getBindingAttr(el, 'type');
    }
    if (!map.type && !typeBinding && map['v-bind']) {
      typeBinding = "(" + (map['v-bind']) + ").type";
    }
```

```
    if (typeBinding) {
      var ifCondition = getAndRemoveAttr(el, 'v-if', true);
      var ifConditionExtra = ifCondition ? ("&&(" + ifCondition + ")") : "";
      var hasElse = getAndRemoveAttr(el, 'v-else', true) != null;
      var elseIfCondition = getAndRemoveAttr(el, 'v-else-if', true);
      // 1. checkbox
      var branch0 = cloneASTElement(el);
      // process for on the main node
      processFor(branch0);
      addRawAttr(branch0, 'type', 'checkbox');
      processElement(branch0, options);
      branch0.processed = true; // prevent it from double-processed
      branch0.if = "(" + typeBinding + ")==='checkbox'" + ifConditionExtra;
      addIfCondition(branch0, {
        exp: branch0.if,
        block: branch0
      });
      // 2. add radio else-if condition
      var branch1 = cloneASTElement(el);
      getAndRemoveAttr(branch1, 'v-for', true);
      addRawAttr(branch1, 'type', 'radio');
      processElement(branch1, options);
      addIfCondition(branch0, {
        exp: "(" + typeBinding + ")==='radio'" + ifConditionExtra,
        block: branch1
      });
      // 3. other
      var branch2 = cloneASTElement(el);
      getAndRemoveAttr(branch2, 'v-for', true);
      addRawAttr(branch2, ':type', typeBinding);
      processElement(branch2, options);
      addIfCondition(branch0, {
        exp: ifCondition,
        block: branch2
      });

      if (hasElse) {
        branch0.else = true;
      } else if (elseIfCondition) {
        branch0.elseif = elseIfCondition;
      }

      return branch0
    }
  }
}

function cloneASTElement (el) {
  return createASTElement(el.tag, el.attrsList.slice(), el.parent)
}

var model$1 = {
  preTransformNode: preTransformNode
};

var modules$1 = [
  klass$1,
  style$1,
  model$1
];
```

```
/*  */

function text (el, dir) {
  if (dir.value) {
    addProp(el, 'textContent', ("_s(" + (dir.value) + ")"), dir);
  }
}

/*  */

function html (el, dir) {
  if (dir.value) {
    addProp(el, 'innerHTML', ("_s(" + (dir.value) + ")"), dir);
  }
}

var directives$1 = {
  model: model,
  text: text,
  html: html
};

/*  */

var baseOptions = {
  expectHTML: true,
  modules: modules$1,
  directives: directives$1,
  isPreTag: isPreTag,
  isUnaryTag: isUnaryTag,
  mustUseProp: mustUseProp,
  canBeLeftOpenTag: canBeLeftOpenTag,
  isReservedTag: isReservedTag,
  getTagNamespace: getTagNamespace,
  staticKeys: genStaticKeys(modules$1)
};

/*  */

var isStaticKey;
var isPlatformReservedTag;

var genStaticKeysCached = cached(genStaticKeys$1);

/**
 * Goal of the optimizer: walk the generated template AST tree
 * and detect sub-trees that are purely static, i.e. parts of
 * the DOM that never needs to change.
 *
 * Once we detect these sub-trees, we can:
 *
 * 1. Hoist them into constants, so that we no longer need to
 *    create fresh nodes for them on each re-render;
 * 2. Completely skip them in the patching process.
 */
function optimize (root, options) {
  if (!root) { return }
  isStaticKey = genStaticKeysCached(options.staticKeys || '');
  isPlatformReservedTag = options.isReservedTag || no;
  // first pass: mark all non-static nodes.
  markStatic$1(root);
  // second pass: mark static roots.
```

```
  markStaticRoots(root, false);
}

function genStaticKeys$1 (keys) {
  return makeMap(
    'type,tag,attrsList,attrsMap,plain,parent,children,attrs,start,end,rawAttrsMap' +
    (keys ? ',' + keys : '')
  )
}

function markStatic$1 (node) {
  node.static = isStatic(node);
  if (node.type === 1) {
    // do not make component slot content static. this avoids
    // 1. components not able to mutate slot nodes
    // 2. static slot content fails for hot-reloading
    if (
      !isPlatformReservedTag(node.tag) &&
      node.tag !== 'slot' &&
      node.attrsMap['inline-template'] == null
    ) {
      return
    }
    for (var i = 0, l = node.children.length; i < l; i++) {
      var child = node.children[i];
      markStatic$1(child);
      if (!child.static) {
        node.static = false;
      }
    }
    if (node.ifConditions) {
      for (var i$1 = 1, l$1 = node.ifConditions.length; i$1 < l$1; i$1++) {
        var block = node.ifConditions[i$1].block;
        markStatic$1(block);
        if (!block.static) {
          node.static = false;
        }
      }
    }
  }
}

function markStaticRoots (node, isInFor) {
  if (node.type === 1) {
    if (node.static || node.once) {
      node.staticInFor = isInFor;
    }
    // For a node to qualify as a static root, it should have children that
    // are not just static text. Otherwise the cost of hoisting out will
    // outweigh the benefits and it's better off to just always render it fresh.
    if (node.static && node.children.length && !(
      node.children.length === 1 &&
      node.children[0].type === 3
    )) {
      node.staticRoot = true;
      return
    } else {
      node.staticRoot = false;
    }
    if (node.children) {
      for (var i = 0, l = node.children.length; i < l; i++) {
        markStaticRoots(node.children[i], isInFor || !!node.for);
```

```
      }
    }
    if (node.ifConditions) {
      for (var i$1 = 1, l$1 = node.ifConditions.length; i$1 < l$1; i$1++) {
        markStaticRoots(node.ifConditions[i$1].block, isInFor);
      }
    }
  }
}

function isStatic (node) {
  if (node.type === 2) { // expression
    return false
  }
  if (node.type === 3) { // text
    return true
  }
  return !!(node.pre || (
    !node.hasBindings && // no dynamic bindings
    !node.if && !node.for && // not v-if or v-for or v-else
    !isBuiltInTag(node.tag) && // not a built-in
    isPlatformReservedTag(node.tag) && // not a component
    !isDirectChildOfTemplateFor(node) &&
    Object.keys(node).every(isStaticKey)
  ))
}

function isDirectChildOfTemplateFor (node) {
  while (node.parent) {
    node = node.parent;
    if (node.tag !== 'template') {
      return false
    }
    if (node.for) {
      return true
    }
  }
  return false
}

/*  */

var fnExpRE = /^([\w$_]+|\([^)]*?\))\s*=>|^function\s*(?:[\w$]+)?\s*\(/;
var fnInvokeRE = /\([^)]*?\);*$/;
var simplePathRE = /^[A-Za-z_$][\w$]*(?:\.[A-Za-z_$][\w$]*|\['[^']*?']|\["[^"]*?"]|\[\d+]|\[[

// KeyboardEvent.keyCode aliases
var keyCodes = {
  esc: 27,
  tab: 9,
  enter: 13,
  space: 32,
  up: 38,
  left: 37,
  right: 39,
  down: 40,
  'delete': [8, 46]
};

// KeyboardEvent.key aliases
var keyNames = {
  // #7880: IE11 and Edge use `Esc` for Escape key name.
```

```
  esc: ['Esc', 'Escape'],
  tab: 'Tab',
  enter: 'Enter',
  // #9112: IE11 uses `Spacebar` for Space key name.
  space: [' ', 'Spacebar'],
  // #7806: IE11 uses key names without `Arrow` prefix for arrow keys.
  up: ['Up', 'ArrowUp'],
  left: ['Left', 'ArrowLeft'],
  right: ['Right', 'ArrowRight'],
  down: ['Down', 'ArrowDown'],
  // #9112: IE11 uses `Del` for Delete key name.
  'delete': ['Backspace', 'Delete', 'Del']
};

// #4868: modifiers that prevent the execution of the listener
// need to explicitly return null so that we can determine whether to remove
// the listener for .once
var genGuard = function (condition) { return ("if(" + condition + ")return null;"); };

var modifierCode = {
  stop: '$event.stopPropagation();',
  prevent: '$event.preventDefault();',
  self: genGuard("$event.target !== $event.currentTarget"),
  ctrl: genGuard("!$event.ctrlKey"),
  shift: genGuard("!$event.shiftKey"),
  alt: genGuard("!$event.altKey"),
  meta: genGuard("!$event.metaKey"),
  left: genGuard("'button' in $event && $event.button !== 0"),
  middle: genGuard("'button' in $event && $event.button !== 1"),
  right: genGuard("'button' in $event && $event.button !== 2")
};

function genHandlers (
  events,
  isNative
) {
  var prefix = isNative ? 'nativeOn:' : 'on:';
  var staticHandlers = "";
  var dynamicHandlers = "";
  for (var name in events) {
    var handlerCode = genHandler(events[name]);
    if (events[name] && events[name].dynamic) {
      dynamicHandlers += name + "," + handlerCode + ",";
    } else {
      staticHandlers += "\"" + name + "\":" + handlerCode + ",";
    }
  }
  staticHandlers = "{" + (staticHandlers.slice(0, -1)) + "}";
  if (dynamicHandlers) {
    return prefix + "_d(" + staticHandlers + ",[" + (dynamicHandlers.slice(0, -1)) + "])"
  } else {
    return prefix + staticHandlers
  }
}

function genHandler (handler) {
  if (!handler) {
    return 'function(){}'
  }

  if (Array.isArray(handler)) {
    return ("[" + (handler.map(function (handler) { return genHandler(handler); }).join(','))
```

```
  }

  var isMethodPath = simplePathRE.test(handler.value);
  var isFunctionExpression = fnExpRE.test(handler.value);
  var isFunctionInvocation = simplePathRE.test(handler.value.replace(fnInvokeRE, ''));

  if (!handler.modifiers) {
    if (isMethodPath || isFunctionExpression) {
      return handler.value
    }
    return ("function($event){" + (isFunctionInvocation ? ("return " + (handler.value)) : har
  } else {
    var code = '';
    var genModifierCode = '';
    var keys = [];
    for (var key in handler.modifiers) {
      if (modifierCode[key]) {
        genModifierCode += modifierCode[key];
        // left/right
        if (keyCodes[key]) {
          keys.push(key);
        }
      } else if (key === 'exact') {
        var modifiers = (handler.modifiers);
        genModifierCode += genGuard(
          ['ctrl', 'shift', 'alt', 'meta']
            .filter(function (keyModifier) { return !modifiers[keyModifier]; })
            .map(function (keyModifier) { return ("$event." + keyModifier + "Key"); })
            .join('||')
        );
      } else {
        keys.push(key);
      }
    }
    if (keys.length) {
      code += genKeyFilter(keys);
    }
    // Make sure modifiers like prevent and stop get executed after key filtering
    if (genModifierCode) {
      code += genModifierCode;
    }
    var handlerCode = isMethodPath
      ? ("return " + (handler.value) + "($event)")
      : isFunctionExpression
        ? ("return (" + (handler.value) + ")($event)")
        : isFunctionInvocation
          ? ("return " + (handler.value))
          : handler.value;
    return ("function($event){" + code + handlerCode + "}")
  }
}

function genKeyFilter (keys) {
  return (
    // make sure the key filters only apply to KeyboardEvents
    // #9441: can't use 'keyCode' in $event because Chrome autofill fires fake
    // key events that do not have keyCode property...
    "if(!$event.type.indexOf('key')&&" +
    (keys.map(genFilterCode).join('&&')) + ")return null;"
  )
}
```

```
function genFilterCode (key) {
  var keyVal = parseInt(key, 10);
  if (keyVal) {
    return ("$event.keyCode!==" + keyVal)
  }
  var keyCode = keyCodes[key];
  var keyName = keyNames[key];
  return (
    "_k($event.keyCode," +
    (JSON.stringify(key)) + "," +
    (JSON.stringify(keyCode)) + "," +
    "$event.key," +
    "" + (JSON.stringify(keyName)) +
    ")"
  )
}

/*  */

function on (el, dir) {
  if (dir.modifiers) {
    warn("v-on without argument does not support modifiers.");
  }
  el.wrapListeners = function (code) { return ("_g(" + code + "," + (dir.value) + ")"); };
}

/*  */

function bind$1 (el, dir) {
  el.wrapData = function (code) {
    return ("_b(" + code + ",'" + (el.tag) + "'," + (dir.value) + "," + (dir.modifiers && dir
  };
}

/*  */

var baseDirectives = {
  on: on,
  bind: bind$1,
  cloak: noop
};

/*  */




var CodegenState = function CodegenState (options) {
  this.options = options;
  this.warn = options.warn || baseWarn;
  this.transforms = pluckModuleFunction(options.modules, 'transformCode');
  this.dataGenFns = pluckModuleFunction(options.modules, 'genData');
  this.directives = extend(extend({}, baseDirectives), options.directives);
  var isReservedTag = options.isReservedTag || no;
  this.maybeComponent = function (el) { return !!el.component || !isReservedTag(el.tag); };
  this.onceId = 0;
  this.staticRenderFns = [];
  this.pre = false;
};
```

```
function generate (
  ast,
  options
) {
  var state = new CodegenState(options);
  var code = ast ? genElement(ast, state) : '_c("div")';
  return {
    render: ("with(this){return " + code + "}"),
    staticRenderFns: state.staticRenderFns
  }
}

function genElement (el, state) {
  if (el.parent) {
    el.pre = el.pre || el.parent.pre;
  }

  if (el.staticRoot && !el.staticProcessed) {
    return genStatic(el, state)
  } else if (el.once && !el.onceProcessed) {
    return genOnce(el, state)
  } else if (el.for && !el.forProcessed) {
    return genFor(el, state)
  } else if (el.if && !el.ifProcessed) {
    return genIf(el, state)
  } else if (el.tag === 'template' && !el.slotTarget && !state.pre) {
    return genChildren(el, state) || 'void 0'
  } else if (el.tag === 'slot') {
    return genSlot(el, state)
  } else {
    // component or element
    var code;
    if (el.component) {
      code = genComponent(el.component, el, state);
    } else {
      var data;
      if (!el.plain || (el.pre && state.maybeComponent(el))) {
        data = genData$2(el, state);
      }

      var children = el.inlineTemplate ? null : genChildren(el, state, true);
      code = "_c('" + (el.tag) + "'" + (data ? ("," + data) : '') + (children ? ("," + childr
    }
    // module transforms
    for (var i = 0; i < state.transforms.length; i++) {
      code = state.transforms[i](el, code);
    }
    return code
  }
}

// hoist static sub-trees out
function genStatic (el, state) {
  el.staticProcessed = true;
  // Some elements (templates) need to behave differently inside of a v-pre
  // node.  All pre nodes are static roots, so we can use this as a location to
  // wrap a state change and reset it upon exiting the pre node.
  var originalPreState = state.pre;
  if (el.pre) {
    state.pre = el.pre;
  }
```

```
  state.staticRenderFns.push(("with(this){return " + (genElement(el, state)) + "}"));
  state.pre = originalPreState;
  return ("_m(" + (state.staticRenderFns.length - 1) + (el.staticInFor ? ',true' : '') + ")")
}

// v-once
function genOnce (el, state) {
  el.onceProcessed = true;
  if (el.if && !el.ifProcessed) {
    return genIf(el, state)
  } else if (el.staticInFor) {
    var key = '';
    var parent = el.parent;
    while (parent) {
      if (parent.for) {
        key = parent.key;
        break
      }
      parent = parent.parent;
    }
    if (!key) {
      state.warn(
        "v-once can only be used inside v-for that is keyed. ",
        el.rawAttrsMap['v-once']
      );
      return genElement(el, state)
    }
    return ("_o(" + (genElement(el, state)) + "," + (state.onceId++) + "," + key + ")")
  } else {
    return genStatic(el, state)
  }
}

function genIf (
  el,
  state,
  altGen,
  altEmpty
) {
  el.ifProcessed = true; // avoid recursion
  return genIfConditions(el.ifConditions.slice(), state, altGen, altEmpty)
}

function genIfConditions (
  conditions,
  state,
  altGen,
  altEmpty
) {
  if (!conditions.length) {
    return altEmpty || '_e()'
  }

  var condition = conditions.shift();
  if (condition.exp) {
    return ("(" + (condition.exp) + ")?" + (genTernaryExp(condition.block)) + ":" + (genIfCor
  } else {
    return ("" + (genTernaryExp(condition.block)))
  }

  // v-if with v-once should generate code like (a)?_m(0):_m(1)
  function genTernaryExp (el) {
```

```
    return altGen
      ? altGen(el, state)
      : el.once
        ? genOnce(el, state)
        : genElement(el, state)
  }
}

function genFor (
  el,
  state,
  altGen,
  altHelper
) {
  var exp = el.for;
  var alias = el.alias;
  var iterator1 = el.iterator1 ? ("," + (el.iterator1)) : '';
  var iterator2 = el.iterator2 ? ("," + (el.iterator2)) : '';

  if (state.maybeComponent(el) &&
    el.tag !== 'slot' &&
    el.tag !== 'template' &&
    !el.key
  ) {
    state.warn(
      "<" + (el.tag) + " v-for=\"" + alias + " in " + exp + "\">: component lists rendered wi
      "v-for should have explicit keys. " +
      "See https://vuejs.org/guide/list.html#key for more info.",
      el.rawAttrsMap['v-for'],
      true /* tip */
    );
  }

  el.forProcessed = true; // avoid recursion
  return (altHelper || '_l') + "((" + exp + ")," +
    "function(" + alias + iterator1 + iterator2 + "){" +
      "return " + ((altGen || genElement)(el, state)) +
    '})'
}

function genData$2 (el, state) {
  var data = '{';

  // directives first.
  // directives may mutate the el's other properties before they are generated.
  var dirs = genDirectives(el, state);
  if (dirs) { data += dirs + ','; }

  // key
  if (el.key) {
    data += "key:" + (el.key) + ",";
  }
  // ref
  if (el.ref) {
    data += "ref:" + (el.ref) + ",";
  }
  if (el.refInFor) {
    data += "refInFor:true,";
  }
  // pre
  if (el.pre) {
    data += "pre:true,";
```

```
  }
  // record original tag name for components using "is" attribute
  if (el.component) {
    data += "tag:\"" + (el.tag) + "\",";
  }
  // module data generation functions
  for (var i = 0; i < state.dataGenFns.length; i++) {
    data += state.dataGenFns[i](el);
  }
  // attributes
  if (el.attrs) {
    data += "attrs:" + (genProps(el.attrs)) + ",";
  }
  // DOM props
  if (el.props) {
    data += "domProps:" + (genProps(el.props)) + ",";
  }
  // event handlers
  if (el.events) {
    data += (genHandlers(el.events, false)) + ",";
  }
  if (el.nativeEvents) {
    data += (genHandlers(el.nativeEvents, true)) + ",";
  }
  // slot target
  // only for non-scoped slots
  if (el.slotTarget && !el.slotScope) {
    data += "slot:" + (el.slotTarget) + ",";
  }
  // scoped slots
  if (el.scopedSlots) {
    data += (genScopedSlots(el, el.scopedSlots, state)) + ",";
  }
  // component v-model
  if (el.model) {
    data += "model:{value:" + (el.model.value) + ",callback:" + (el.model.callback) + ",expre
  }
  // inline-template
  if (el.inlineTemplate) {
    var inlineTemplate = genInlineTemplate(el, state);
    if (inlineTemplate) {
      data += inlineTemplate + ",";
    }
  }
  data = data.replace(/,$/, '') + '}';
  // v-bind dynamic argument wrap
  // v-bind with dynamic arguments must be applied using the same v-bind object
  // merge helper so that class/style/mustUseProp attrs are handled correctly.
  if (el.dynamicAttrs) {
    data = "_b(" + data + ",\"" + (el.tag) + "\"," + (genProps(el.dynamicAttrs)) + ")";
  }
  // v-bind data wrap
  if (el.wrapData) {
    data = el.wrapData(data);
  }
  // v-on data wrap
  if (el.wrapListeners) {
    data = el.wrapListeners(data);
  }
  return data
}
```

```
function genDirectives (el, state) {
  var dirs = el.directives;
  if (!dirs) { return }
  var res = 'directives:[';
  var hasRuntime = false;
  var i, l, dir, needRuntime;
  for (i = 0, l = dirs.length; i < l; i++) {
    dir = dirs[i];
    needRuntime = true;
    var gen = state.directives[dir.name];
    if (gen) {
      // compile-time directive that manipulates AST.
      // returns true if it also needs a runtime counterpart.
      needRuntime = !!gen(el, dir, state.warn);
    }
    if (needRuntime) {
      hasRuntime = true;
      res += "{name:\"" + (dir.name) + "\",rawName:\"" + (dir.rawName) + "\"" + (dir.value ?
    }
  }
  if (hasRuntime) {
    return res.slice(0, -1) + ']'
  }
}

function genInlineTemplate (el, state) {
  var ast = el.children[0];
  if (el.children.length !== 1 || ast.type !== 1) {
    state.warn(
      'Inline-template components must have exactly one child element.',
      { start: el.start }
    );
  }
  if (ast && ast.type === 1) {
    var inlineRenderFns = generate(ast, state.options);
    return ("inlineTemplate:{render:function(){" + (inlineRenderFns.render) + "},staticRender
  }
}

function genScopedSlots (
  el,
  slots,
  state
) {
  // by default scoped slots are considered "stable", this allows child
  // components with only scoped slots to skip forced updates from parent.
  // but in some cases we have to bail-out of this optimization
  // for example if the slot contains dynamic names, has v-if or v-for on them...
  var needsForceUpdate = el.for || Object.keys(slots).some(function (key) {
    var slot = slots[key];
    return (
      slot.slotTargetDynamic ||
      slot.if ||
      slot.for ||
      containsSlotChild(slot) // is passing down slot from parent which may be dynamic
    )
  });

  // #9534: if a component with scoped slots is inside a conditional branch,
  // it's possible for the same component to be reused but with different
  // compiled slot content. To avoid that, we generate a unique key based on
  // the generated code of all the slot contents.
```

```
  var needsKey = !!el.if;

  // OR when it is inside another scoped slot or v-for (the reactivity may be
  // disconnected due to the intermediate scope variable)
  // #9438, #9506
  // TODO: this can be further optimized by properly analyzing in-scope bindings
  // and skip force updating ones that do not actually use scope variables.
  if (!needsForceUpdate) {
    var parent = el.parent;
    while (parent) {
      if (
        (parent.slotScope && parent.slotScope !== emptySlotScopeToken) ||
        parent.for
      ) {
        needsForceUpdate = true;
        break
      }
      if (parent.if) {
        needsKey = true;
      }
      parent = parent.parent;
    }
  }

  var generatedSlots = Object.keys(slots)
    .map(function (key) { return genScopedSlot(slots[key], state); })
    .join(',');

  return ("scopedSlots:_u([" + generatedSlots + "]" + (needsForceUpdate ? ",null,true" : "")
}

function hash(str) {
  var hash = 5381;
  var i = str.length;
  while(i) {
    hash = (hash * 33) ^ str.charCodeAt(--i);
  }
  return hash >>> 0
}

function containsSlotChild (el) {
  if (el.type === 1) {
    if (el.tag === 'slot') {
      return true
    }
    return el.children.some(containsSlotChild)
  }
  return false
}

function genScopedSlot (
  el,
  state
) {
  var isLegacySyntax = el.attrsMap['slot-scope'];
  if (el.if && !el.ifProcessed && !isLegacySyntax) {
    return genIf(el, state, genScopedSlot, "null")
  }
  if (el.for && !el.forProcessed) {
    return genFor(el, state, genScopedSlot)
  }
  var slotScope = el.slotScope === emptySlotScopeToken
```

```
    ? ""
    : String(el.slotScope);
  var fn = "function(" + slotScope + "){" +
    "return " + (el.tag === 'template'
      ? el.if && isLegacySyntax
        ? ("(" + (el.if) + ")?" + (genChildren(el, state) || 'undefined') + ":undefined")
        : genChildren(el, state) || 'undefined'
      : genElement(el, state)) + "}";
  // reverse proxy v-slot without scope on this.$slots
  var reverseProxy = slotScope ? "" : ",proxy:true";
  return ("{key:" + (el.slotTarget || "\"default\"") + ",fn:" + fn + reverseProxy + "}")
}

function genChildren (
  el,
  state,
  checkSkip,
  altGenElement,
  altGenNode
) {
  var children = el.children;
  if (children.length) {
    var el$1 = children[0];
    // optimize single v-for
    if (children.length === 1 &&
      el$1.for &&
      el$1.tag !== 'template' &&
      el$1.tag !== 'slot'
    ) {
      var normalizationType = checkSkip
        ? state.maybeComponent(el$1) ? ",1" : ",0"
        : "";
      return ("" + ((altGenElement || genElement)(el$1, state)) + normalizationType)
    }
    var normalizationType$1 = checkSkip
      ? getNormalizationType(children, state.maybeComponent)
      : 0;
    var gen = altGenNode || genNode;
    return ("[" + (children.map(function (c) { return gen(c, state); }).join(',')) + "]" + (r
  }
}

// determine the normalization needed for the children array.
// 0: no normalization needed
// 1: simple normalization needed (possible 1-level deep nested array)
// 2: full normalization needed
function getNormalizationType (
  children,
  maybeComponent
) {
  var res = 0;
  for (var i = 0; i < children.length; i++) {
    var el = children[i];
    if (el.type !== 1) {
      continue
    }
    if (needsNormalization(el) ||
        (el.ifConditions && el.ifConditions.some(function (c) { return needsNormalization(c.b
      res = 2;
      break
    }
    if (maybeComponent(el) ||
```

```
        (el.ifConditions && el.ifConditions.some(function (c) { return maybeComponent(c.block
      res = 1;
    }
  }
  return res
}

function needsNormalization (el) {
  return el.for !== undefined || el.tag === 'template' || el.tag === 'slot'
}

function genNode (node, state) {
  if (node.type === 1) {
    return genElement(node, state)
  } else if (node.type === 3 && node.isComment) {
    return genComment(node)
  } else {
    return genText(node)
  }
}

function genText (text) {
  return ("_v(" + (text.type === 2
    ? text.expression // no need for () because already wrapped in _s()
    : transformSpecialNewlines(JSON.stringify(text.text))) + ")")
}

function genComment (comment) {
  return ("_e(" + (JSON.stringify(comment.text)) + ")")
}

function genSlot (el, state) {
  var slotName = el.slotName || '"default"';
  var children = genChildren(el, state);
  var res = "_t(" + slotName + (children ? ("," + children) : '');
  var attrs = el.attrs || el.dynamicAttrs
    ? genProps((el.attrs || []).concat(el.dynamicAttrs || []).map(function (attr) { return ({
        // slot props are camelized
        name: camelize(attr.name),
        value: attr.value,
        dynamic: attr.dynamic
      }); }))
    : null;
  var bind$$1 = el.attrsMap['v-bind'];
  if ((attrs || bind$$1) && !children) {
    res += ",null";
  }
  if (attrs) {
    res += "," + attrs;
  }
  if (bind$$1) {
    res += (attrs ? '' : ',null') + "," + bind$$1;
  }
  return res + ')'
}

// componentName is el.component, take it as argument to shun flow's pessimistic refinement
function genComponent (
  componentName,
  el,
  state
) {
```

```
  var children = el.inlineTemplate ? null : genChildren(el, state, true);
  return ("_c(" + componentName + "," + (genData$2(el, state)) + (children ? ("," + children)
}

function genProps (props) {
  var staticProps = "";
  var dynamicProps = "";
  for (var i = 0; i < props.length; i++) {
    var prop = props[i];
    var value = transformSpecialNewlines(prop.value);
    if (prop.dynamic) {
      dynamicProps += (prop.name) + "," + value + ",";
    } else {
      staticProps += "\"" + (prop.name) + "\":" + value + ",";
    }
  }
  staticProps = "{" + (staticProps.slice(0, -1)) + "}";
  if (dynamicProps) {
    return ("_d(" + staticProps + ",[" + (dynamicProps.slice(0, -1)) + "])")
  } else {
    return staticProps
  }
}

// #3895, #4268
function transformSpecialNewlines (text) {
  return text
    .replace(/\u2028/g, '\\u2028')
    .replace(/\u2029/g, '\\u2029')
}

/*  */



// these keywords should not appear inside expressions, but operators like
// typeof, instanceof and in are allowed
var prohibitedKeywordRE = new RegExp('\\b' + (
  'do,if,for,let,new,try,var,case,else,with,await,break,catch,class,const,' +
  'super,throw,while,yield,delete,export,import,return,switch,default,' +
  'extends,finally,continue,debugger,function,arguments'
).split(',').join('\\b|\\b') + '\\b');

// these unary operators should not be used as property/method names
var unaryOperatorsRE = new RegExp('\\b' + (
  'delete,typeof,void'
).split(',').join('\\s*\\([^\\)]*\\)|\\b') + '\\s*\\([^\\)]*\\)');

// strip strings in expressions
var stripStringRE = /'(?:[^'\\]|\\.)*'|"(?:[^"\\]|\\.)*"|`(?:[^`\\]|\\.)*\$\{|\}(?:[^`\\]|\\.

// detect problematic expressions in a template
function detectErrors (ast, warn) {
  if (ast) {
    checkNode(ast, warn);
  }
}

function checkNode (node, warn) {
  if (node.type === 1) {
    for (var name in node.attrsMap) {
      if (dirRE.test(name)) {
```

```
        var value = node.attrsMap[name];
        if (value) {
          var range = node.rawAttrsMap[name];
          if (name === 'v-for') {
            checkFor(node, ("v-for=\"" + value + "\""), warn, range);
          } else if (onRE.test(name)) {
            checkEvent(value, (name + "=\"" + value + "\""), warn, range);
          } else {
            checkExpression(value, (name + "=\"" + value + "\""), warn, range);
          }
        }
      }
    }
    if (node.children) {
      for (var i = 0; i < node.children.length; i++) {
        checkNode(node.children[i], warn);
      }
    }
  } else if (node.type === 2) {
    checkExpression(node.expression, node.text, warn, node);
  }
}

function checkEvent (exp, text, warn, range) {
  var stipped = exp.replace(stripStringRE, '');
  var keywordMatch = stipped.match(unaryOperatorsRE);
  if (keywordMatch && stipped.charAt(keywordMatch.index - 1) !== '$') {
    warn(
      "avoid using JavaScript unary operator as property name: " +
      "\"" + (keywordMatch[0]) + "\" in expression " + (text.trim()),
      range
    );
  }
  checkExpression(exp, text, warn, range);
}

function checkFor (node, text, warn, range) {
  checkExpression(node.for || '', text, warn, range);
  checkIdentifier(node.alias, 'v-for alias', text, warn, range);
  checkIdentifier(node.iterator1, 'v-for iterator', text, warn, range);
  checkIdentifier(node.iterator2, 'v-for iterator', text, warn, range);
}

function checkIdentifier (
  ident,
  type,
  text,
  warn,
  range
) {
  if (typeof ident === 'string') {
    try {
      new Function(("var " + ident + "=_"));
    } catch (e) {
      warn(("invalid " + type + " \"" + ident + "\" in expression: " + (text.trim())), range)
    }
  }
}

function checkExpression (exp, text, warn, range) {
  try {
    new Function(("return " + exp));
```

```
  } catch (e) {
    var keywordMatch = exp.replace(stripStringRE, '').match(prohibitedKeywordRE);
    if (keywordMatch) {
      warn(
        "avoid using JavaScript keyword as property name: " +
        "\"" + (keywordMatch[0]) + "\"\n  Raw expression: " + (text.trim()),
        range
      );
    } else {
      warn(
        "invalid expression: " + (e.message) + " in\n\n" +
        "    " + exp + "\n\n" +
        "  Raw expression: " + (text.trim()) + "\n",
        range
      );
    }
  }
}

/*  */

var range = 2;

function generateCodeFrame (
  source,
  start,
  end
) {
  if ( start === void 0 ) start = 0;
  if ( end === void 0 ) end = source.length;

  var lines = source.split(/\r?\n/);
  var count = 0;
  var res = [];
  for (var i = 0; i < lines.length; i++) {
    count += lines[i].length + 1;
    if (count >= start) {
      for (var j = i - range; j <= i + range || end > count; j++) {
        if (j < 0 || j >= lines.length) { continue }
        res.push(("" + (j + 1) + (repeat$1(" ", 3 - String(j + 1).length)) + "|  " + (lines[j
        var lineLength = lines[j].length;
        if (j === i) {
          // push underline
          var pad = start - (count - lineLength) + 1;
          var length = end > count ? lineLength - pad : end - start;
          res.push("   |  " + repeat$1(" ", pad) + repeat$1("^", length));
        } else if (j > i) {
          if (end > count) {
            var length$1 = Math.min(end - count, lineLength);
            res.push("   |  " + repeat$1("^", length$1));
          }
          count += lineLength + 1;
        }
      }
      break
    }
  }
  return res.join('\n')
}

function repeat$1 (str, n) {
  var result = '';
```

```
  if (n > 0) {
    while (true) { // eslint-disable-line
      if (n & 1) { result += str; }
      n >>>= 1;
      if (n <= 0) { break }
      str += str;
    }
  }
  return result
}

/*  */



function createFunction (code, errors) {
  try {
    return new Function(code)
  } catch (err) {
    errors.push({ err: err, code: code });
    return noop
  }
}

function createCompileToFunctionFn (compile) {
  var cache = Object.create(null);

  return function compileToFunctions (
    template,
    options,
    vm
  ) {
    options = extend({}, options);
    var warn$$1 = options.warn || warn;
    delete options.warn;

    /* istanbul ignore if */
    {
      // detect possible CSP restriction
      try {
        new Function('return 1');
      } catch (e) {
        if (e.toString().match(/unsafe-eval|CSP/)) {
          warn$$1(
            'It seems you are using the standalone build of Vue.js in an ' +
            'environment with Content Security Policy that prohibits unsafe-eval. ' +
            'The template compiler cannot work in this environment. Consider ' +
            'relaxing the policy to allow unsafe-eval or pre-compiling your ' +
            'templates into render functions.'
          );
        }
      }
    }

    // check cache
    var key = options.delimiters
      ? String(options.delimiters) + template
      : template;
    if (cache[key]) {
      return cache[key]
    }
```

```
    // compile
    var compiled = compile(template, options);

    // check compilation errors/tips
    {
      if (compiled.errors && compiled.errors.length) {
        if (options.outputSourceRange) {
          compiled.errors.forEach(function (e) {
            warn$$1(
              "Error compiling template:\n\n" + (e.msg) + "\n\n" +
              generateCodeFrame(template, e.start, e.end),
              vm
            );
          });
        } else {
          warn$$1(
            "Error compiling template:\n\n" + template + "\n\n" +
            compiled.errors.map(function (e) { return ("- " + e); }).join('\n') + '\n',
            vm
          );
        }
      }
      if (compiled.tips && compiled.tips.length) {
        if (options.outputSourceRange) {
          compiled.tips.forEach(function (e) { return tip(e.msg, vm); });
        } else {
          compiled.tips.forEach(function (msg) { return tip(msg, vm); });
        }
      }
    }

    // turn code into functions
    var res = {};
    var fnGenErrors = [];
    res.render = createFunction(compiled.render, fnGenErrors);
    res.staticRenderFns = compiled.staticRenderFns.map(function (code) {
      return createFunction(code, fnGenErrors)
    });

    // check function generation errors.
    // this should only happen if there is a bug in the compiler itself.
    // mostly for codegen development use
    /* istanbul ignore if */
    {
      if ((!compiled.errors || !compiled.errors.length) && fnGenErrors.length) {
        warn$$1(
          "Failed to generate render function:\n\n" +
          fnGenErrors.map(function (ref) {
            var err = ref.err;
            var code = ref.code;

            return ((err.toString()) + " in\n\n" + code + "\n");
          }).join('\n'),
          vm
        );
      }
    }

    return (cache[key] = res)
  }
}
```

```
/*  */

function createCompilerCreator (baseCompile) {
  return function createCompiler (baseOptions) {
    function compile (
      template,
      options
    ) {
      var finalOptions = Object.create(baseOptions);
      var errors = [];
      var tips = [];

      var warn = function (msg, range, tip) {
        (tip ? tips : errors).push(msg);
      };

      if (options) {
        if (options.outputSourceRange) {
          // $flow-disable-line
          var leadingSpaceLength = template.match(/^\s*/)[0].length;

          warn = function (msg, range, tip) {
            var data = { msg: msg };
            if (range) {
              if (range.start != null) {
                data.start = range.start + leadingSpaceLength;
              }
              if (range.end != null) {
                data.end = range.end + leadingSpaceLength;
              }
            }
            (tip ? tips : errors).push(data);
          };
        }
        // merge custom modules
        if (options.modules) {
          finalOptions.modules =
            (baseOptions.modules || []).concat(options.modules);
        }
        // merge custom directives
        if (options.directives) {
          finalOptions.directives = extend(
            Object.create(baseOptions.directives || null),
            options.directives
          );
        }
        // copy other options
        for (var key in options) {
          if (key !== 'modules' && key !== 'directives') {
            finalOptions[key] = options[key];
          }
        }
      }

      finalOptions.warn = warn;

      var compiled = baseCompile(template.trim(), finalOptions);
      {
        detectErrors(compiled.ast, warn);
      }
      compiled.errors = errors;
      compiled.tips = tips;
```

```
      return compiled
    }

    return {
      compile: compile,
      compileToFunctions: createCompileToFunctionFn(compile)
    }
  }
}

/*  */

// `createCompilerCreator` allows creating compilers that use alternative
// parser/optimizer/codegen, e.g the SSR optimizing compiler.
// Here we just export a default compiler using the default parts.
var createCompiler = createCompilerCreator(function baseCompile (
  template,
  options
) {
  var ast = parse(template.trim(), options);
  if (options.optimize !== false) {
    optimize(ast, options);
  }
  var code = generate(ast, options);
  return {
    ast: ast,
    render: code.render,
    staticRenderFns: code.staticRenderFns
  }
});

/*  */

var ref$1 = createCompiler(baseOptions);
var compile = ref$1.compile;
var compileToFunctions = ref$1.compileToFunctions;

/*  */

// check whether current browser encodes a char inside attribute values
var div;
function getShouldDecode (href) {
  div = div || document.createElement('div');
  div.innerHTML = href ? "<a href=\"\n\"/>" : "<div a=\"\n\"/>";
  return div.innerHTML.indexOf('&#10;') > 0
}

// #3663: IE encodes newlines inside attribute values while other browsers don't
var shouldDecodeNewlines = inBrowser ? getShouldDecode(false) : false;
// #6828: chrome encodes content in a[href]
var shouldDecodeNewlinesForHref = inBrowser ? getShouldDecode(true) : false;

/*  */

var idToTemplate = cached(function (id) {
  var el = query(id);
  return el && el.innerHTML
});

var mount = Vue.prototype.$mount;
Vue.prototype.$mount = function (
  el,
```

```
  hydrating
) {
  el = el && query(el);

  /* istanbul ignore if */
  if (el === document.body || el === document.documentElement) {
    warn(
      "Do not mount Vue to <html> or <body> - mount to normal elements instead."
    );
    return this
  }

  var options = this.$options;
  // resolve template/el and convert to render function
  if (!options.render) {
    var template = options.template;
    if (template) {
      if (typeof template === 'string') {
        if (template.charAt(0) === '#') {
          template = idToTemplate(template);
          /* istanbul ignore if */
          if (!template) {
            warn(
              ("Template element not found or is empty: " + (options.template)),
              this
            );
          }
        }
      } else if (template.nodeType) {
        template = template.innerHTML;
      } else {
        {
          warn('invalid template option:' + template, this);
        }
        return this
      }
    } else if (el) {
      template = getOuterHTML(el);
    }
    if (template) {
      /* istanbul ignore if */
      if (config.performance && mark) {
        mark('compile');
      }

      var ref = compileToFunctions(template, {
        outputSourceRange: "development" !== 'production',
        shouldDecodeNewlines: shouldDecodeNewlines,
        shouldDecodeNewlinesForHref: shouldDecodeNewlinesForHref,
        delimiters: options.delimiters,
        comments: options.comments
      }, this);
      var render = ref.render;
      var staticRenderFns = ref.staticRenderFns;
      options.render = render;
      options.staticRenderFns = staticRenderFns;

      /* istanbul ignore if */
      if (config.performance && mark) {
        mark('compile end');
        measure(("vue " + (this._name) + " compile"), 'compile', 'compile end');
      }
```

```
    }
  }
  return mount.call(this, el, hydrating)
};

/**
 * Get outerHTML of elements, taking care
 * of SVG elements in IE as well.
 */
function getOuterHTML (el) {
  if (el.outerHTML) {
    return el.outerHTML
  } else {
    var container = document.createElement('div');
    container.appendChild(el.cloneNode(true));
    return container.innerHTML
  }
}

Vue.compile = compileToFunctions;

module.exports = Vue;
```

Step 2:

Now let's update `components/MessageList.vue` to use it:

components/MessageList.vue

```
<template>
  <ul>
    <Message :message="message" v-for="message in messages"/>
  </ul>
</template>

<script>
import Message from "./Message";

export default {
  props: ["messages"],
  components: {
    Message
  }
};
</script>
```

Now that we are done with testing two components as a unit, let's move on to the next phase of testing.